Mathematics of Neural Networks

Volume V: Large-Scale Models, Emergence and Frontiers

Scaling laws, mechanistic interpretability, and open research frontiers

---

Every derivation step by step

Every symbol defined

Explicit numerical examples

Geometric intuitions

Silvano Salvador

Contents

53.1 The Quadratic Bottleneck **2**

53.2 Sliding Window Attention **3**

53.3 Grouped Query Attention (GQA) **3**

53.4 Mixture-of-Experts (MoE) **4**

54.1 Rotary Positional Embedding (RoPE) **6**

54.2 YaRN: Context Extension via Frequency Interpolation **7**

54.3 ALiBi: Attention with Linear Biases 8

55.1 Linear State Space Models **9**

55.2 S4: Structured State Spaces **10**

55.3 Mamba: Selective State Spaces **11**

56.1 Linear Attention **12**

56.2 Equivalence Theorem **12**

57.1 Empirical Scaling Laws **13**

57.2 Deriving the Power Laws **13**

57.3 Optimal Compute Allocation (Chinchilla Scaling) **13**

58.1 Formalising Emergence **15**

58.2 Double Descent **15**

58.3 In-Context Learning as Implicit Bayesian Inference **15**

59.1 The Computational Complexity View **17**

59.2 Chain-of-Thought Provides Serial Depth **17**

59.3 Scratchpad as Working Memory **17**

60.1 Dense Passage Retrieval (DPR) **18**

60.2 Maximum Inner-Product Search (MIPS) **18**

60.3 RAG-Fusion **18**

61.1 The Superposition Hypothesis **20**

61.2 Sparse Autoencoders (SAE) **20**

61.3 Circuit Discovery: Indirect Object Identification **20**

62.1 Bradley-Terry Preference Model **22**

62.2 PPO for Language Models 22
62.3 Constitutional AI (CAI) 22
63.1 Leaky Integrate-and-Fire (LIF) Neuron 23
63.2 Qubit Hilbert Space 23
63.3 Variational Quantum Circuits (VQC) 24
64.1 The NAS Problem 25
64.2 DARTS: Continuous Relaxation 25
64.3 Deriving the Final Architecture 25
65.1 Autoregressive Factorisation 26
65.2 MADE: Masked Autoencoder for Distribution Estimation 26
65.3 PixelCNN: Causal Convolutions 26
65.4 VQ-VAE: Vector Quantised Variational Autoencoder 27
65.5 Synthesis: The Five-Volume Arc 28
53.5 Random Feature Maps and FAVOR+ 29
53.6 Hyper-Attention 30
54.4 T5 Relative Position Bias 31
54.5 ALiBi as an Exponential Decay Kernel 31
54.6 xPos: Extended Positional Embeddings 32
55.4 HiPPO Optimality Theorem 33
55.5 S5: Multi-Input Multi-Output SSM 33
55.6 Mamba-2 and State Space Duality (SSD) 33
57.4 Inference-Time Compute Scaling 35
57.5 Data Mixture Optimisation 35
57.6 The Compute-Optimal Pareto Frontier 36
58.4 The Grokking Phenomenon 37
58.5 Modular Arithmetic as a Mechanistic Case Study 37
59.4 Tree-of-Thought (ToT) 39
59.5 Self-Consistency as Majority Voting 39

59.6 AlphaProof and Lean Formal Verification 40
61.4 Optimal Superposition Geometry 41
61.5 Polysemanticity and Neuron Geometry 41
61.6 TopK SAEs 42
62.4 KL-Reward Tradeoff and Overoptimisation 43
62.5 SimPO and Reference-Free Alignment 43
62.6 ORPO: Odds-Ratio Preference Optimisation 44
63.4 Barren Plateaus in VQCs 45
63.5 Neuromorphic Hardware: Intel Loihi-2 45
64.4 Hardware-Aware NAS 46
64.5 Once-for-All (OFA) 46
65.6 VQ-VAE-2: Hierarchical Discrete Latents 48
65.7 Consistency Models 48
65.8 Latent Diffusion Models (LDMs) 49
66.1 Intrinsic Dimension of Fine-Tuning 50
66.2 LoRA: Low-Rank Adaptation 50
66.3 QLoRA: 4-bit Quantisation with LoRA 51
66.4 Test-Time Training (TTT) 51
67.1 CLIP: Contrastive Language-Image Pre-training 53
67.2 Vision Transformer (ViT) 53
67.3 Flamingo: Vision-Language Models via Cross-Attention 54
68.1 Score Matching 55
68.2 DDPM Derivation 55
68.3 Conditional Flow Matching 56
69.1 The Implicit Bias of Gradient Descent 57
69.2 Benign Overfitting 57
69.3 Equivariance as a Design Principle 58
69.4 The Depth Separation Problem 58

69.5 Foundation Models and Emergent World Models 59
53.7 Multi-Head Latent Attention (MLA) 62
53.8 Sparse Attention: Block-Sparse Patterns 62
53.9 KV Cache Compression via Streaming 63
54.7 Lost in the Middle: Attention Sink in Long Context 64
54.8 RULER: Robust Context Understanding Evaluation 64
54.9 Positional Generalisation: The ROPE Frequency Spectrum 65
55.7 RWKV Architecture 66
55.8 Griffin: Recurrent Blocks with Gated Linear Recurrence 67
56.3 Attention as Nadaraya-Watson Estimation 68
56.4 Universal Approximation for Transformers 68
57.7 Best-of-N Sampling with Verifiers 70
57.8 Process Reward Models (PRMs) 70
57.9 Reasoning Tokens and the Extended Chinchilla Model 71
58.6 The Metric-Artefact Hypothesis 72
58.7 Circuit Formation and Grokking Phase Diagram 72
60.4 ColBERT Late Interaction 74
61.7 Induction Heads 75
61.8 Virtual Weights and Composition 76
62.7 GRPO: Group Relative Policy Optimisation 77
62.8 Reward Shaping for Mathematical Reasoning 78
66.5 Adapter Tuning 79
66.6 Prefix Tuning 79
68.4 The SDE Framework 81
68.5 Consistency Between SDE Formulations 82
70.1 The Speculative Decoding Protocol 85
70.2 Expected Speedup Analysis 86
70.3 Self-Speculative Decoding 86

71.1 Mixture of Depths (MoD) 87
71.2 Early Exit 87
71.3 Adaptive Computation Time (ACT) 88
72.1 Hinton's Knowledge Distillation 89
72.2 Feature-Level Distillation 89
72.3 Structured Pruning 90
73.1 Catastrophic Forgetting 91
73.2 Elastic Weight Consolidation (EWC) 91
73.3 Gradient Projection 92
74.1 Tokenisation as Data Compression 93
74.2 Byte Pair Encoding (BPE) 93
74.3 Unigram Language Model Tokeniser 93
74.4 Vocabulary Size and Model Capacity 94
75.1 The Three Axes of Modern Neural Network Mathematics 95
75.2 Unifying Formula: The Efficiency-Quality-Alignment Triad 95
75.3 Open Mathematical Questions 96
53.10 MoE as Optimal Transport 97
53.11 Expert Specialisation at Equilibrium 97
55.9 Frequency Response of SSMs 99
55.10 Audio and Video Modelling with SSMs 99
57.10 Exact Parameter Count 101
57.11 FLOP Analysis 101
76.1 Calibration 103
76.2 Temperature Scaling 103
76.3 Out-of-Distribution Detection 104
77.1 Message Passing Neural Networks 105
77.2 The Weisfeiler-Leman Bound 105
77.3 Graph Transformers 106

78.1 Floating-Point Arithmetic for Deep Learning 107
78.2 BF16 Advantages 107
78.3 FP8 Training 107
79.1 Transformer Expressivity 109
79.2 Identifiability of Attention Patterns 109
79.3 Mean-Field Analysis of Attention Training 110
80.1 Vision-Language Alignment 111
80.2 Patch Token Sequences and 2D Positional Structure 111
80.3 Cross-Modal Consistency 112
81.1 Greedy Decoding and Beam Search 113
81.2 Nucleus Sampling 113
81.3 Repetition Penalties 114
82.1 Minimal Transformer (d=4, 1 head, 2 layers) 115
82.2 Attention Pattern Interpretation 116
83.1 The Constitutional AI Framework 119
83.2 Iterative Self-Improvement 119
83.3 Red-Teaming as Adversarial Optimisation 120
84.1 The Green-Red Watermarking Scheme 121
84.2 Statistical Detection 121
84.3 Quality Degradation 122
85.1 SwiGLU Derivation 123
85.2 Activation Function Comparisons 123
85.3 Neural Tangent Kernel for Different Activations 123
86.1 RASP: Restricted Access Sequence Processing 125
86.2 Formal Language Classes 125
86.3 Implications for In-Context Learning 126
87.1 The Instruction Tuning Objective 127
87.2 Task Embeddings and Clustering 127

87.3 Chain of Templates and Prompt Robustness 128

88.1 Functional Framework for Position Encodings 129

88.2 Relative Position Encodings 129

88.3 RoPE: Full Derivation from Requirements 130

88.4 Comparison: Extrapolation Properties 130

Appendix A: Notation and Symbols 131

Appendix B: Key Derivations 133

Appendix D: Glossary of Algorithms 135

Appendix E: Complexity Reference Table 137

Bibliography 138

PART 53

Efficient Attention: Sliding Window, GQA and Mixture-of-Experts

Standard self-attention scales as $O(n^2 d)$ in sequence length n. We derive sliding window attention, grouped query attention, and the conditional computation mathematics of Mixture-of-Experts layers.

53.1 The Quadratic Bottleneck

Standard self-attention for a sequence of n tokens with dimension d:

$$\mathbf{A} = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)\mathbf{V}, \qquad \mathbf{Q},\mathbf{K},\mathbf{V} \in \mathbb{R}^{n\times d_k}$$

The matrix $\mathbf{Q}\mathbf{K}^\top \in \mathbb{R}^{n\times n}$ requires $O(n^2)$ memory and $O(n^2 d_k)$ FLOPs. For $n = 128$K tokens, this is ≈ 16GB just for the attention matrix in float32.

Why $O(n^2)$ is a hard barrier for naive attention. Each of the n query vectors must compute an inner product with each of the n key vectors. This is n^2 dot products, each costing $O(d_k)$ multiply-adds. The resulting $n \times n$ score matrix must be stored before softmax and then multiplied by $\mathbf{V}$. No algebraic trick reduces this without approximation — the quadratic cost is intrinsic to all-pairs similarity computation.

FlashAttention: IO-aware exact computation. Dao et al. (2022) observe that the bottleneck is not FLOPs but memory bandwidth. The $n \times n$ matrix cannot fit in GPU SRAM (typically 20–80 MB) for $n > 8192$. FlashAttention tiles the computation: it processes blocks of B_Q queries against blocks of B_{KV} keys at a time, keeping running statistics for the softmax denominator (online softmax).

The key identity enabling online softmax: for log-sum-exp $m^{(j)} = \max_{i\le j} q_i k_i^\top$:

$$\text{softmax}(qk^\top)_{1:j} = \frac{e^{qk_{1:j}^\top - m^{(j)}}}{\sum_{i=1}^{j} e^{qk_i^\top - m^{(j)}}}$$

This can be updated incrementally as new keys arrive, without storing the full $n \times n$ matrix. FlashAttention reads/writes $O(n)$ data to HBM (vs. $O(n^2)$), achieving 2–4× wall-clock speedup despite identical FLOPs.

EXAMPLE — Memory comparison for $n = 32768$, $d_k = 128$, $H = 32$, fp16

Standard attention matrix: $32768^2 \times 32 \times 2\text{B} = 68.7$ GB per layer.
FlashAttention SRAM usage (block size $B = 64$): $2 \times 64 \times 128 \times 2\text{B} = 32$ KB per block.
Speedup on A100 (HBM bandwidth 2 TB/s, SRAM 19 TB/s): $\sim 3\times$ end-to-end for forward + backward.

FlashAttention-2 improvements. Dao (2023) further reduces non-matmul FLOPs, parallelises over the sequence dimension (not just batch/heads), and improves the backward pass. FlashAttention-3 (Shah et al. 2024) exploits the H100's asynchronous memory copy engines to overlap GEMM and softmax operations, achieving $\sim 75\%$ utilisation of peak FLOP throughput.

53.2 Sliding Window Attention

Instead of attending to all n tokens, each token attends only to a local window of w tokens:

$$A_{ij} = \begin{cases} \frac{\exp(q_i k_j^\top / \sqrt{d_k})}{\sum_{|i'-j|\le w/2} \exp(q_i k_{i'}^\top / \sqrt{d_k})} & \text{if } |i-j| \le w/2 \\ 0 & \text{otherwise} \end{cases}$$

Complexity: $O(nw)$ — linear in n for fixed window w. Longformer and BigBird combine sliding window attention (for local context) with global tokens (for long-range dependencies).

Receptive field. With L layers of window w, a token's effective receptive field is Lw tokens. For $L = 12$, $w = 512$: receptive field of 6144 tokens.

Dilated sliding window. To increase the receptive field without increasing the window size, attention can skip positions:

$$A_{ij} \neq 0 \iff |i-j| \in \{0, d, 2d, \dots, (w/2) \cdot d\}$$

for dilation factor d. With $w = 512$ and $d = 4$: receptive field $512 \times 4 = 2048$ tokens at the same $O(nw)$ cost. Different layers can use different dilation factors to capture multi-scale structure.

Global tokens in Longformer. Selected tokens (e.g. `[CLS]`, task-specific tokens) attend to all n positions and are attended to by all positions:

- Global token g computes $a_{gj} = q_g^\top k_j / \sqrt{d_k}$ for all $j \in \{1, \dots, n\}$.
- All tokens attend to g: $a_{ig} = q_i^\top k_g / \sqrt{d_k}$.

With $G \ll n$ global tokens, the total cost is $O(nw + nG) = O(n(w+G))$, still linear.

Example — Longformer on a 4096-token document, $w = 512$, $G = 8$

Local attention: $4096 \times 512 = 2{,}097{,}152$ attention operations.
Global attention: $4096 \times 8 \times 2 = 65{,}536$ operations.
Standard full attention: $4096^2 = 16{,}777{,}216$ operations.
Speedup factor: $\approx 7.6\times$. For $n = 128$K: local $128\text{K} \times 512$, standard 128K^2; speedup $\approx 250\times$.

BigBird: random attention. BigBird adds r random attention edges per token in addition to sliding window and global tokens. Formally the attention graph is:

$$\mathcal{G} = \mathcal{G}_{\text{window}} \cup \mathcal{G}_{\text{global}} \cup \mathcal{G}_{\text{random}}$$

The Expander Graph theorem guarantees that a random graph with $O(\log n)$ edges per node has diameter $O(\log n)$, ensuring that information from any token can reach any other in $O(\log n)$ layers even without global tokens. BigBird is thus theoretically Turing-complete: it can simulate any sequential computation, unlike standard attention-only models with fixed context.

53.3 Grouped Query Attention (GQA)

Standard multi-head attention uses H query, H key, and H value heads. During inference, the KV cache stores all past key-value pairs: memory $O(2 \cdot n \cdot H \cdot d_k)$.

Multi-query attention (MQA). Share one key and one value head across all H query heads: H queries, 1 key, 1 value. KV cache reduced by $H\times$.

Grouped query attention (GQA). Intermediate: G key-value groups, H/G query heads per group.

GQA Projection

- $\mathbf{Q} = \mathbf{x}\mathbf{W}_Q \in \mathbb{R}^{n\times Hd_k}$ (H query heads, as before)
- $\mathbf{K} = \mathbf{x}\mathbf{W}_K \in \mathbb{R}^{n\times Gd_k}$ ($G \ll H$ key heads)
- $\mathbf{V} = \mathbf{x}\mathbf{W}_V \in \mathbb{R}^{n\times Gd_k}$ (G value heads)

Query head h uses key/value head $g = \lfloor hG/H \rfloor$.

Example — GQA KV cache comparison, $H = 32$, $G = 8$, $n = 4096$, $d_k = 128$

Standard MHA: $2 \times 4096 \times 32 \times 128 = 33.5$M values.
GQA ($G = 8$): $2 \times 4096 \times 8 \times 128 = 8.4$M values. $4\times$ reduction.
MQA ($G = 1$): $2 \times 4096 \times 1 \times 128 = 1.0$M values. $32\times$ reduction.

Quality-memory tradeoff. MQA degrades quality on knowledge-intensive tasks (TriviaQA, NaturalQuestions) by approximately 1–3 points compared to MHA. GQA with $G = H/4$ recovers most of this loss while retaining most of the KV cache savings. The intuition: different query heads attending to the same key-value head lose the ability to attend to diverse aspects of the context simultaneously. With $G > 1$ groups, each group of queries can still specialise.

Uptraining existing MHA models to GQA. Ainslie et al. (2023) propose initialising the G key-value heads by mean-pooling the original H MHA key-value heads within each group:

$$\mathbf{W}_{K,g}^{\text{GQA}} = \frac{G}{H} \sum_{h:\lfloor hG/H \rfloor = g} \mathbf{W}_{K,h}^{\text{MHA}}$$

Fine-tuning on a small fraction ($\approx 5\%$) of the original training data then recovers full quality. This avoids training GQA models from scratch.

Flash Attention and memory bandwidth. The memory bandwidth cost of the KV cache dominates inference latency for autoregressive generation at batch size 1. With GQA ($G = 8$, $H = 32$), the KV cache bandwidth required per generated token is reduced $4\times$. For a 70B parameter Llama-2 model with $n = 4096$ context, $H = 64$, $G = 8$, $d_k = 128$:

$$\text{KV cache per layer} = 2 \times 4096 \times 8 \times 128 \times 2 \text{ bytes} = 16.8 \text{ MB per layer}$$

With 80 layers: 1.34 GB, versus 10.7 GB for standard MHA. At memory bandwidth 3.35 TB/s (H100), reading the KV cache takes 0.4 ms per token — enabling $\approx$ 2500 tokens/second generation speed.

53.4 Mixture-of-Experts (MoE)

A MoE layer replaces a single FFN with E expert FFNs, routing each token to the top-k experts:

$$\text{MoE}(\mathbf{x}) = \sum_{e \in \text{Top}_k(\mathbf{g}(\mathbf{x}))} g_e(\mathbf{x}) \cdot \text{FFN}_e(\mathbf{x})$$

$$g_e(\mathbf{x}) = \frac{\exp(\mathbf{x}\mathbf{W}_g)_e + \epsilon_e}{\sum_e \exp(\mathbf{x}\mathbf{W}_g)_e + \epsilon_e}$$

where $\epsilon_e \sim \mathcal{N}(0, 1/E^2)$ is noise for load balancing.

Conditional computation. Only k/E fraction of parameters are active per token. Mixtral 8×7B has 47B total parameters but uses only 12.9B per token (top-2 of 8 experts): better quality than a dense 12.9B model at similar inference cost.

Auxiliary load-balancing loss.

$$\mathcal{L}_{\text{aux}} = \alpha \cdot E \sum_{e=1}^{E} f_e \cdot p_e$$

where f_e = fraction of tokens assigned to expert e, p_e = average routing probability to expert e. This penalises unequal expert utilisation.

Expert routing as a discrete optimisation problem. The Top-k selection is not differentiable, so gradients cannot flow through the routing decision. The gating weights g_e for *selected* experts receive gradients; unselected experts receive no gradient from this token. This sparse gradient flow means each expert is updated only by the tokens routed to it, creating a form of specialisation.

Expert collapse. Without load balancing, routing collapses: a few experts receive all tokens (the rich-get-richer effect of softmax), while most experts receive near-zero gradient and remain untrained. The auxiliary loss prevents this by penalising the correlation $f_e \cdot p_e$:

- If expert e receives many tokens (f_e high) *and* has high routing probability (p_e high), the term $f_e p_e$ is large, adding to the loss.
- This encourages the router to spread tokens uniformly ($f_e \approx 1/E$ for all e).

Expert parallelism. MoE models distribute experts across devices. Each device holds E/D experts (for D devices). During the forward pass, after routing, tokens must be *dispatched* to the correct device, processed, and results *gathered* back. This all-to-all communication is the main overhead of MoE at large scale.

Example — Expert dispatch for $E = 8$, $k = 2$, batch size $B = 512$, sequence length $n = 1024$

Total tokens: $512 \times 1024 = 524{,}288$. Each assigned to 2 experts.
Expected tokens per expert: $524{,}288 \times 2/8 = 131{,}072$.
Expert buffer capacity: typically $1.25 \times 131{,}072 = 163{,}840$ tokens per expert.
Tokens routed to a full expert (above capacity) are dropped. At $\alpha = 10^{-2}$, expert utilisation $> 95\%$ with $< 0.1\%$ token drop rate.

Expert Choice routing. Lepikhim et al. (2022) reverse the routing: instead of tokens choosing experts, each expert chooses its top-c tokens. This guarantees perfect load balance by construction but breaks the causal structure needed for autoregressive generation (an expert may select tokens from different positions, leaking future information). Expert Choice is thus restricted to encoder architectures or non-causal settings.

PART 54

Modern LLM Architectures: Gemma, RoPE and YaRN

We derive rotary positional embeddings from first principles, show how they enable length extrapolation, and analyse the YaRN context extension method via interpolation in the frequency domain.

54.1 Rotary Positional Embedding (RoPE)

RoPE Definition

For position m and dimension index $i \in \{0, \dots, d/2-1\}$, the rotation angle is $\theta_i = m/10000^{2i/d}$. The rotary embedding applies a block-diagonal rotation to query and key vectors:

$$\mathbf{R}_m \mathbf{q} = \begin{pmatrix} q_1 \cos\theta_0 - q_2 \sin\theta_0 \\ q_1 \sin\theta_0 + q_2 \cos\theta_0 \\ q_3 \cos\theta_1 - q_4 \sin\theta_1 \\ q_3 \sin\theta_1 + q_4 \cos\theta_1 \\ \vdots \end{pmatrix}$$

Key property. The inner product of rotated query and key depends only on the *relative* position $m-n$:

$$(\mathbf{R}_m \mathbf{q})^\top (\mathbf{R}_n \mathbf{k}) = \mathbf{q}^\top \mathbf{R}_{m-n}^\top \mathbf{k} = \mathbf{q}^\top \mathbf{R}_{n-m} \mathbf{k}$$

Proof. For a single 2D block:

$$\begin{aligned}(\mathbf{R}_m \mathbf{q})_{\{2i,2i+1\}}^\top (\mathbf{R}_n \mathbf{k})_{\{2i,2i+1\}} &= (q_{2i}\cos m\theta_i - q_{2i+1}\sin m\theta_i)(k_{2i}\cos n\theta_i - k_{2i+1}\sin n\theta_i) \\ &\quad + (q_{2i}\sin m\theta_i + q_{2i+1}\cos m\theta_i)(k_{2i}\sin n\theta_i + k_{2i+1}\cos n\theta_i)\end{aligned}$$

Expanding and collecting: $= q_{2i}k_{2i}\cos(m-n)\theta_i + q_{2i+1}k_{2i+1}\cos(m-n)\theta_i + \text{cross terms}$.

This is exactly $(\mathbf{R}_{m-n}\mathbf{q})_{\{2i,2i+1\}}^\top \mathbf{k}_{\{2i,2i+1\}}$. $\square$

Efficient implementation. Rather than constructing the $d \times d$ block-diagonal rotation matrix explicitly, RoPE is applied element-wise. Given $\mathbf{q} \in \mathbb{R}^d$, reshape to pairs (q_{2i}, q_{2i+1}) and apply:

$$\begin{aligned}\text{RoPE}(\mathbf{q}, m)_{2i} &= q_{2i}\cos(m\theta_i) - q_{2i+1}\sin(m\theta_i) \\ \text{RoPE}(\mathbf{q}, m)_{2i+1} &= q_{2i}\sin(m\theta_i) + q_{2i+1}\cos(m\theta_i)\end{aligned}$$

This is equivalent to complex multiplication: $(q_{2i} + iq_{2i+1}) \cdot e^{im\theta_i}$. In code, this is a single element-wise multiply (with precomputed cos, sin tables) plus a "rotate" operation, costing $O(d)$ with no matrix construction.

NTK-aware interpolation. A key insight from "NTK-by-parts" (bloc97, 2023): the base frequency $b = 10000$ sets the wavelength $\lambda_i = 2\pi b^{2i/d}$. Scaling the base to $b' = b \cdot s^{d/(d-2)}$ for scale factor s effectively rescales all wavelengths uniformly. This is equivalent to linear interpolation in the angular domain. For $s = 8$: $b' = 10000 \cdot 8^{64/62} \approx 87{,}000$. Models with this modified base can handle $8\times$ their training context with minimal perplexity degradation, often *without fine-tuning*.

Example — RoPE on a 4-dimensional model ($d = 4$, $b = 10000$)

Dimensions: pairs $(0, 1)$ and $(2, 3)$. Frequencies: $\theta_0 = 1/10000^0 = 1.0$, $\theta_1 = 1/10000^{2/4} = 0.01$.
For position $m = 5$:

- Pair $(0, 1)$: rotate by $5 \times 1.0 = 5$ radians. $\cos(5) \approx 0.284$, $\sin(5) \approx -0.959$.
- Pair $(2, 3)$: rotate by $5 \times 0.01 = 0.05$ radians. $\cos(0.05) \approx 0.999$, $\sin(0.05) \approx 0.050$.

The slow rotation of pair $(2, 3)$ provides long-range positional signal (distinguishes positions 0 vs. 1000); the fast rotation of pair $(0, 1)$ provides short-range signal (distinguishes consecutive positions).

54.2 YaRN: Context Extension via Frequency Interpolation

At training context length L_{train}, frequencies θ_i are in the range $[10000^{-2i/d} L_{\text{train}}, 10000^{-2i/d}]$. At test length $L_{\text{test}} > L_{\text{train}}$, naïve positional encoding extrapolates — high-frequency dimensions rotate too fast, degrading performance.

YaRN interpolation. Scale the effective base $b = 10000$:

$$\theta_i^{\text{new}} = \frac{m}{b'^{2i/d}}, \qquad b' = b \cdot \left(\frac{L_{\text{test}}/L_{\text{train}} - 1}{\alpha} \right)^{\beta}$$

YaRN uses a *ramp function* $\gamma(r)$ that blends:

- Low-frequency dimensions ($r < r_{\text{min}}$): position interpolation (scale positions down by $L_{\text{test}}/L_{\text{train}}$).
- High-frequency dimensions ($r > r_{\text{max}}$): no modification (already short-range).
- Intermediate: smooth blend.

This allows extending from $L_{\text{train}} = 4096$ to $L_{\text{test}} = 128\text{K}$ with minimal fine-tuning (1000 steps vs. full retraining).

The wavelength perspective. Each dimension i of RoPE oscillates with a spatial wavelength:

$$\lambda_i = 2\pi \cdot b^{2i/d}$$

For $b = 10000$, $d = 128$: $\lambda_0 = 2\pi$ (one cycle per 6.28 tokens) up to $\lambda_{63} = 2\pi \cdot 10000 = 62{,}832$ tokens. At $L_{\text{train}} = 4096$, dimensions with $\lambda_i > L_{\text{train}}$ have never completed a full rotation during training — these are the low-frequency dimensions that can be linearly interpolated.

Position interpolation (Su et al. 2023). The simplest approach: rescale positions by $L_{\text{train}}/L_{\text{test}}$, so positions $0, 1, \ldots, L_{\text{test}} - 1$ are mapped to $0, L_{\text{train}}/L_{\text{test}}, \ldots, L_{\text{train}} - 1$. The issue: high-frequency dimensions are now compressed into a very small range, reducing their positional discriminability.

YaRN Ramp Function

Define the interpolation factor per dimension:

$$\gamma(r) = \begin{cases} 0 & r < r_{\text{min}} \\ \frac{r - r_{\text{min}}}{r_{\text{max}} - r_{\text{min}}} & r_{\text{min}} \le r \le r_{\text{max}} \\ 1 & r > r_{\text{max}} \end{cases}$$

where $r = \log(\lambda_i / L_{\text{train}}) / \log(L_{\text{train}})$ is the normalised log-wavelength.
The effective position for dimension i:

$$\tilde{m}_i = \left[(1 - \gamma(r_i)) \frac{L_{\text{train}}}{L_{\text{test}}} + \gamma(r_i) \right] \cdot m$$

High-frequency dimensions ($\gamma = 1$): no scaling. Low-frequency dimensions ($\gamma = 0$): full linear interpolation.

Example — YaRN: $L_{\text{train}} = 4096$, $L_{\text{test}} = 32768$, $d = 128$, $r_{\text{min}} = 0.06$, $r_{\text{max}} = 0.1$

Scale factor: $s = L_{\text{test}} / L_{\text{train}} = 8$.
Dimension $i = 63$ (lowest frequency): $\lambda_{63} \approx 62832$. $r_{63} \approx 0.012 < r_{\text{min}}$. $\gamma = 0$: full interpolation. Position $m = 10000$ becomes $m/8 = 1250$.
Dimension $i = 0$ (highest frequency): $\lambda_0 = 2\pi \approx 6.28$. $r_0 \approx -0.15 < r_{\text{min}}$ as well... actually high-frequency means *small* r. YaRN's parameterisation ensures that very high-frequency dims ($r > r_{\text{max}}$, meaning $\lambda_i \gg L_{\text{train}}$) receive no interpolation.
Result: PPL increases by $< 5\%$ at 32K context after 1000 steps fine-tuning, vs. $> 200\%$ increase for naive extrapolation.

54.3 ALiBi: Attention with Linear Biases

ALiBi (Press et al. 2022) takes a different approach: instead of modifying positional encodings, it adds a linear bias to attention scores:

$$a_{ij} = \frac{q_i^\top k_j}{\sqrt{d_k}} - m_h \cdot |i - j|$$

where m_h is a head-specific slope. For H heads, slopes are set to $\{2^{-8/H}, 2^{-16/H}, \ldots, 2^{-8}\}$.

ALiBi length generalisation. Because ALiBi only penalises distance (not uses absolute positions), models trained on length L_{train} can be evaluated at $L_{\text{test}} > L_{\text{train}}$ with graceful degradation — no fine-tuning required. The penalty $m_h \cdot |i - j|$ grows with distance, but the model has learned to compensate for distances up to L_{train}. Remarkably, ALiBi generalises to $L_{\text{test}} = 4L_{\text{train}}$ with minimal PPL increase, unlike RoPE without interpolation.

Part 55

State Space Models: S4 and Mamba

State space models (SSMs) combine the efficiency of convolutions with the sequential expressivity of RNNs. We derive the S4 parameterisation via diagonal-plus-low-rank structure and the selective scan of Mamba.

55.1 Linear State Space Models

A continuous-time linear SSM maps input $u(t) \in \mathbb{R}$ to output $y(t) \in \mathbb{R}$ via:

$$\dot{\mathbf{h}}(t) = \mathbf{A}\mathbf{h}(t) + \mathbf{B}u(t), \qquad y(t) = \mathbf{C}\mathbf{h}(t) + Du(t)$$

where $\mathbf{A} \in \mathbb{R}^{N\times N}$, $\mathbf{B} \in \mathbb{R}^{N\times 1}$, $\mathbf{C} \in \mathbb{R}^{1\times N}$, $D \in \mathbb{R}$.

Discretisation. Using the zero-order hold (ZOH) with step Δ:

$$\bar{\mathbf{A}} = e^{\Delta\mathbf{A}}, \qquad \bar{\mathbf{B}} = (\Delta\mathbf{A})^{-1}(e^{\Delta\mathbf{A}} - \mathbf{I})\Delta\mathbf{B}$$

Discrete-time recurrence: $\mathbf{h}_k = \bar{\mathbf{A}}\mathbf{h}_{k-1} + \bar{\mathbf{B}}u_k$, $y_k = \mathbf{C}\mathbf{h}_k$.

Transfer function and frequency response. The SSM is a linear time-invariant (LTI) system. Its transfer function in the z-domain is:

$$H(z) = \mathbf{C}(z\mathbf{I} - \bar{\mathbf{A}})^{-1}\bar{\mathbf{B}} + D$$

The frequency response $H(e^{i\omega})$ for $\omega \in [-\pi, \pi]$ determines which frequencies the SSM amplifies or attenuates. This is not a property of attention-based models, making SSMs interpretable from a signal-processing perspective.

HiPPO: Optimal polynomial projection. A key challenge in SSMs is choosing $\mathbf{A}$. HiPPO (Gu et al. 2020) derives $\mathbf{A}$ analytically as the matrix that optimally compresses the input history $u(t')$ for $t' \in (-\infty, t]$ into N coefficients of a polynomial approximation. For Legendre polynomials and a sliding window measure, HiPPO gives:

$$A_{nk} = \begin{cases} -(2n+1)^{1/2}(2k+1)^{1/2} & n > k \\ n+1 & n = k \\ 0 & n < k \end{cases}$$

This triangular structure allows the state $\mathbf{h}(t)$ to represent the coefficients of the best degree-N polynomial approximation to the recent input history, enabling long-range memory compression.

Example — HiPPO memory with $N = 4$

State $\mathbf{h}(t) = (h_0, h_1, h_2, h_3)$ represents the projection of input history onto Legendre polynomials P_0, P_1, P_2, P_3 on $[0, t]$.
For a step function $u(t) = \mathbf{1}[t \geq t_0]$: the HiPPO state tracks the fraction of the window that has been in the "on" state. As $t \to \infty$, $h_0 \to 1$ (the average), $h_1, h_2, h_3 \to 0$ (the derivative structure decays), capturing the long-run mean with minimal representation error.

55.2 S4: Structured State Spaces

The bottleneck in SSMs is computing powers of $\mathbf{A}$. S4 uses the *DPLR* (Diagonal Plus Low-Rank) parameterisation:

$$\mathbf{A} = \square - \mathbf{P}\mathbf{Q}^*$$

where $\square = \mathrm{diag}(\lambda_1, \dots, \lambda_N)$ is diagonal and $\mathbf{P}, \mathbf{Q} \in \mathbb{C}^{N \times r}$ is low-rank.

Efficient convolution kernel. The SSM is equivalent to a convolution with kernel:

$$K = (\mathbf{C}\bar{\mathbf{B}},\ \mathbf{C}\bar{\mathbf{A}}\bar{\mathbf{B}},\ \mathbf{C}\bar{\mathbf{A}}^2\bar{\mathbf{B}},\ \dots,\ \mathbf{C}\bar{\mathbf{A}}^{L-1}\bar{\mathbf{B}})$$

With DPLR structure, K can be computed via the Cauchy kernel in $O(N + L)$ using the generating function:

$$\hat{K}(z) = \mathbf{C}(z\mathbf{I} - \bar{\mathbf{A}})^{-1}\bar{\mathbf{B}} + D$$

Full convolution: $O(L \log L)$ via FFT. Training is thus $O(L \log L)$ vs. $O(L^2)$ for attention.

Why DPLR enables efficient computation. For a general $N \times N$ matrix $\mathbf{A}$, computing $e^{\Delta \mathbf{A}}$ requires $O(N^3)$ work (matrix exponentiation). With DPLR, the Woodbury identity allows:

$$(z\mathbf{I} - \bar{\mathbf{A}})^{-1} = (z\mathbf{I} - \bar{\square} + \bar{\mathbf{P}}\bar{\mathbf{Q}}^*)^{-1} = (z\mathbf{I} - \bar{\square})^{-1} - (z\mathbf{I} - \bar{\square})^{-1}\bar{\mathbf{P}}\left[\mathbf{I} + \bar{\mathbf{Q}}^*(z\mathbf{I} - \bar{\square})^{-1}\bar{\mathbf{P}}\right]^{-1}\bar{\mathbf{Q}}^*(z\mathbf{I} - \bar{\square})^{-1}$$

Since $\bar{\square}$ is diagonal, $(z\mathbf{I} - \bar{\square})^{-1}$ is diagonal with entries $1/(z - \lambda_k)$. The remaining computation involves a $r \times r$ inverse (cheap for small r) and products of N-vectors. The kernel $\hat{K}(z) = \mathbf{C}(z\mathbf{I} - \bar{\mathbf{A}})^{-1}\bar{\mathbf{B}}$ reduces to a sum of N Cauchy terms:

$$\hat{K}(z) = \sum_{k=1}^{N} \frac{c_k}{z - \lambda_k}$$

where c_k depends on $\mathbf{B}, \mathbf{C}$. Evaluating this at L points $z_1, \dots, z_L$ (the DFT frequencies) requires $O(NL)$ operations, then the FFT convolution is $O(L \log L)$.

Example — S4 convolution kernel computation, $N = 64$, $L = 4096$

DPLR setup: $\square \in \mathbb{C}^{64}$ (64 diagonal entries), rank-1 correction ($r = 1$).
Kernel computation:

1. Compute $c_k = \mathbf{C}_k\bar{\mathbf{B}}_k/(z - \lambda_k)$ at $L = 4096$ DFT frequencies: $64 \times 4096 = 262{,}144$ operations.
2. FFT the kernel (length 4096): $4096 \log_2(4096) \approx 49{,}152$ operations.
3. FFT the input: $49{,}152$ operations.
4. Pointwise multiply: 4096 operations.
5. Inverse FFT: $49{,}152$ operations.

Total: $\approx 410{,}000$ operations, vs. $4096^2 = 16{,}777{,}216$ for dense convolution.

S4D: Diagonal S4. Gu et al. (2022) showed that a purely diagonal $\mathbf{A}$ (dropping the low-rank correction) achieves competitive performance while simplifying implementation. The diagonal entries are initialised to $\lambda_k = -1/2 + i\pi k$ (imaginary part ensures oscillatory behaviour for sequence modelling). S4D eliminates the Woodbury inversion entirely, reducing to N independent scalar SSMs.

55.3 Mamba: Selective State Spaces

S4 is linear time-invariant: $\mathbf{A}, \mathbf{B}, \mathbf{C}$ do not depend on the input. Mamba (Gu & Dao, 2023) makes them input-dependent:

$$\Delta_k = \text{softplus}(\text{Linear}(u_k)), \quad \mathbf{B}_k = \text{Linear}(u_k), \quad \mathbf{C}_k = \text{Linear}(u_k)$$

Selective scan. Input-dependence breaks the static convolution kernel: the recurrence must be computed sequentially. Mamba implements this efficiently on GPU via a *parallel scan algorithm* (prefix sum generalisation):

For $\mathbf{h}_k = a_k \mathbf{h}_{k-1} + b_k$, the prefix scan computes all $\mathbf{h}_k$ in $O(\log L)$ parallel steps using the associativity of the operation $(a, b) \otimes (c, d) = (ac, ad + b)$.

Example — Parallel scan for $L = 4$

Values $(a_1, b_1), (a_2, b_2), (a_3, b_3), (a_4, b_4)$.
Level 1 (pairs): compute $(a_2 a_1, a_2 b_1 + b_2)$ and $(a_4 a_3, a_4 b_3 + b_4)$.
Level 2: combine to get $h_4 = a_4 a_3 a_2 a_1 h_0 + a_4 a_3 a_2 b_1 + a_4 a_3 b_2 + a_4 b_3 + b_4$.
All 4 states computed in $\lceil \log_2 4 \rceil = 2$ parallel steps.

PART 56

SSM vs Attention Duality

Recent work reveals a precise mathematical duality between linear attention and SSMs. We derive the linear attention kernel trick, show the equivalence, and discuss what each perspective reveals.

56.1 Linear Attention

Standard attention: $\text{softmax}(\mathbf{Q}\mathbf{K}^\top/\sqrt{d})\mathbf{V}$. The softmax is nonlinear; the matrix $\mathbf{Q}\mathbf{K}^\top$ is $O(n^2)$.

Kernel trick. Replace softmax with a positive kernel: $\text{Attn}(\mathbf{Q},\mathbf{K},\mathbf{V}) = \frac{\phi(\mathbf{Q})[\phi(\mathbf{K})^\top\mathbf{V}]}{\phi(\mathbf{Q})[\phi(\mathbf{K})^\top\mathbf{1}]}$.

Using the associativity of matrix products:

$$\phi(\mathbf{Q})[\phi(\mathbf{K})^\top\mathbf{V}] = \phi(\mathbf{Q})\sum_{i=1}^{n}\phi(\mathbf{k}_i)\mathbf{v}_i^\top$$

The cumulative sum $\mathbf{S}_j = \sum_{i=1}^{j}\phi(\mathbf{k}_i)\mathbf{v}_i^\top \in \mathbb{R}^{d\times d}$ can be updated recurrently:

$$\mathbf{S}_j = \mathbf{S}_{j-1} + \phi(\mathbf{k}_j)\mathbf{v}_j^\top, \qquad \mathbf{o}_j = \frac{\phi(\mathbf{q}_j)^\top\mathbf{S}_j}{\phi(\mathbf{q}_j)^\top\sum_{i\le j}\phi(\mathbf{k}_i)}$$

This is a *linear recurrence* with state $\mathbf{S}_j$ — exactly the SSM recurrence with $N = d^2$ state dimensions!

56.2 Equivalence Theorem

Linear Attention = SSM (up to parameterisation)

Any linear attention model with kernel ϕ is equivalent to an SSM with:

$$\mathbf{A} = \mathbf{I}, \quad \mathbf{B} = \phi(\mathbf{k}_j)^\top, \quad \mathbf{C} = \phi(\mathbf{q}_j), \quad \text{state} = \mathbf{S}_j$$

Conversely, any diagonal SSM with $\mathbf{A} = \mathbf{I}$ corresponds to linear attention with a specific kernel.

Implications. Selective SSMs (Mamba) correspond to *input-dependent* attention patterns. The gating in Mamba controls which past context is retained in the state — equivalent to a learned, position-dependent decay of attention scores.

PART 57

Chinchilla Scaling Laws: Derivation from Statistical Mechanics

Kaplan et al. (2020) and Hoffmann et al. (2022) empirically discovered power laws for language model loss. We derive these from an idealised bias-variance model and analyse the optimal compute allocation.

57.1 Empirical Scaling Laws

The neural language model cross-entropy loss L as a function of model size N (parameters) and dataset size D (tokens) follows:

$$L(N, D) = E + \frac{A}{N^\alpha} + \frac{B}{D^\beta}$$

where $E = H(p_{\text{data}})$ is the irreducible entropy, and Chinchilla found $\alpha \approx 0.34$, $\beta \approx 0.28$.

57.2 Deriving the Power Laws

Bias term A/N^α: approximation error. A network of size N approximates the true function in a function class of complexity $\sim N$. For smooth functions in $\mathbb{R}^d$ approximated by a neural network with N parameters, approximation theory gives error $\sim N^{-\alpha/d}$ for α depending on smoothness. Natural language functions are not in $\mathbb{R}^d$ but analogous arguments suggest polynomial decay.

Variance term B/D^β: estimation error. With D training examples and a model of effective complexity C, the estimation error scales as C/D (statistical learning theory). Since $C \sim N$ and we observe $L \sim N/D^1$ for fixed N, the empirical $\beta \approx 0.28$ suggests sublinear complexity growth.

57.3 Optimal Compute Allocation (Chinchilla Scaling)

For a fixed compute budget $C = 6ND$ FLOPs (the factor 6 accounting for forward + backward + optimizer), we minimise $L(N, D)$ subject to $ND = C/6$.

Chinchilla Optimal Allocation

Minimise $L(N, C/(6N))$ over N:

$$\frac{d}{dN} L = -\frac{A\alpha}{N^{\alpha+1}} + \frac{B\beta}{(C/6N)^\beta} \cdot \frac{C/6}{N^2} = 0$$

Solving: $N^* \propto C^a$, $D^* \propto C^b$ where $a = \frac{\beta}{\alpha+\beta}$, $b = \frac{\alpha}{\alpha+\beta}$.
With $\alpha = 0.34$, $\beta = 0.28$: $a \approx 0.45$, $b \approx 0.55$. Optimal: $D^* \approx 20N^*$ tokens per parameter.

EXAMPLE — Optimal compute allocation for 10^{23} FLOPs

$C = 10^{23}$, $6ND = C \Rightarrow ND = 1.67 \times 10^{22}$.
$N^* \approx (1.67 \times 10^{22})^{0.45} \approx 6.7 \times 10^9$ (6.7B parameters).
$D^* = 1.67 \times 10^{22} / 6.7 \times 10^9 \approx 2.5 \times 10^{12}$ (2.5T tokens).

GPT-4 ($\approx 10^{25}$ FLOPs) is optimally trained with ≈ 270B params and ≈ 50T tokens — consistent with reported training configurations.

PART 58

Emergence and Phase Transitions in Neural Networks

Emergent capabilities appear abruptly as models scale. We formalise emergence using phase transitions from statistical physics, analyse the double descent phenomenon, and discuss sharp thresholds.

58.1 Formalising Emergence

Emergence (Ganguli et al.)

A capability $\mathcal{C}$ is *emergent* at scale N^* if:

- For $N < N^*$: model performance on $\mathcal{C}$ is at chance ($\approx$ random baseline).
- For $N \geq N^*$: performance increases rapidly above chance.

The sharpness of the transition is the hallmark of emergence.

Statistical physics analogy. A ferromagnet undergoes a phase transition at critical temperature T_c: magnetisation $m(T) \propto (T_c - T)^\beta$ for $T < T_c$, zero above. Emergent capabilities behave analogously: Performance $\propto (N - N^*)^\gamma$ for $N > N^*$.

58.2 Double Descent

Classical bias-variance theory predicts a U-shaped test error curve. Modern overparameterised models exhibit a *double descent*:

1. **Classical regime** ($N < D$): test error decreases, then increases (overfitting).
2. **Interpolation threshold** ($N \approx D$): test error peaks (the model just barely interpolates the data, with high variance).
3. **Modern regime** ($N \gg D$): test error decreases again as overparameterisation enables implicit regularisation.

Random matrix theory explanation. At $N/D \to 1$ (aspect ratio $\to 1$), the Hessian of the empirical risk has eigenvalues spreading according to the Marchenko-Pastur distribution, with minimum eigenvalue $\lambda_{\min} \to 0$. The condition number $\kappa = \lambda_{\max}/\lambda_{\min} \to \infty$: the model is ill-conditioned, causing high-variance predictions. For $N/D \gg 1$, the minimum eigenvalue lifts away from zero.

58.3 In-Context Learning as Implicit Bayesian Inference

GPT-style models perform in-context learning (ICL): given examples $(x_1, y_1), \ldots, (x_k, y_k)$ in the prompt, they predict y_{k+1} for x_{k+1}.

Bayesian ICL Hypothesis (Xie et al. 2021)

Pretraining on data from a mixture of tasks $\{p_\theta\}_\theta$ (with prior $p(\theta)$) induces a posterior:

$$p(\theta | x_1, y_1, \ldots, x_k, y_k) \propto p(\theta) \prod_{i=1}^{k} p_\theta(x_i, y_i)$$

The LLM's prediction is:

$$p_{\text{LLM}}(y_{k+1}|x_{k+1}, \text{context}) \approx \int p_\theta(y_{k+1}|x_{k+1})\, p(\theta|\text{context})\, d\theta$$

This is exactly Bayesian model averaging over the task distribution.

Implication. ICL is not mere pattern matching but approximates Bayesian inference implicitly in the forward pass of the transformer. The number of in-context examples k plays the role of data for posterior concentration: more examples $\Rightarrow$ sharper posterior $\Rightarrow$ better task identification.

PART 59

Chain-of-Thought as Formal Program Execution

We formalise chain-of-thought reasoning as scratchpad computation in a formal language, analyse the complexity advantage it provides, and connect it to circuit complexity lower bounds.

59.1 The Computational Complexity View

A transformer with L layers, H heads, and d dimensions computes a function $f : \Sigma^n \to \Sigma^m$. Without chain-of-thought, the transformer's output is a function of the input alone, computed in L parallel rounds.

TC^0 Circuit Class

The class TC^0 contains problems solvable by constant-depth, polynomial-size circuits with threshold gates. Transformers (for fixed L) compute functions in TC^0.

Problem. Many reasoning tasks (arithmetic, logical deduction) require $\Omega(\log n)$ depth (not in TC^0 unless $\mathsf{TC}^0 = \mathsf{NC}^1$, which is open but believed false). Fixed-depth transformers provably cannot solve n-digit multiplication without exponentially large width (Merrill & Sabharwal, 2023).

59.2 Chain-of-Thought Provides Serial Depth

With chain-of-thought, the model generates T intermediate tokens before the final answer. Each token generation is one transformer forward pass (L layers deep). The total effective computation depth is $L \cdot T$.

Theorem (informal). A transformer with L layers and chain-of-thought of length $T = \Omega(n)$ can simulate any Boolean circuit of depth $O(n)$, enabling it to solve tasks in NC^1 (and potentially P).

EXAMPLE — Multi-digit addition without vs. with CoT

Without CoT. Model must produce "1234 + 5678 = 6912" in one forward pass. At n-digit precision, requires exponential width to carry-propagate in $O(1)$ depth.
With CoT. Model writes: "units: 4+8=12, write 2 carry 1; tens: 3+7+1=11, write 1 carry 1; ...; answer: 6912."
Each digit processed in one generation step: $O(n)$ steps, linear depth.

59.3 Scratchpad as Working Memory

Formally, CoT defines a *scratchpad language* $\mathcal{S}$ and a transition function $\tau : (\text{input}, \text{scratchpad state}) \to (\text{next token})$. This is a $\mathcal{S}$-augmented transformer, equivalent to a nondeterministic Turing machine with T steps of computation. The scratchpad compensates for the transformer's lack of persistent state between positions.

PART 60

Retrieval-Augmented Generation: Dense Retrieval and RAG-Fusion

RAG separates parametric knowledge (model weights) from non-parametric knowledge (retrieved documents). We derive dense passage retrieval (DPR), the maximum inner-product search problem, and RAG-Fusion re-ranking.

60.1 Dense Passage Retrieval (DPR)

DPR Architecture

Two BERT encoders: query encoder E_Q, passage encoder E_P.

$$\text{sim}(q, p) = E_Q(q)^\top E_P(p) \qquad \in \mathbb{R}$$

The retriever returns the top-k passages: $\arg\max_{p \in \mathcal{C}} E_Q(q)^\top E_P(p)$.

Training. Trained with in-batch negatives: for a batch of B query-passage pairs $\{(q_i, p_i^+)\}$, the positive passage p_i^+ must score higher than all other passages in the batch. Loss for query i:

$$\mathcal{L}_i = -\log \frac{e^{E_Q(q_i)^\top E_P(p_i^+)}}{\sum_{j=1}^{B} e^{E_Q(q_i)^\top E_P(p_j^+)}}$$

This is cross-entropy over B candidates — equivalent to InfoNCE with $B - 1$ negatives.

60.2 Maximum Inner-Product Search (MIPS)

Exact MIPS is $O(|\mathcal{C}|)$ per query. For $|\mathcal{C}| = 10^7$ passages and $d = 768$ dimensions, this requires 7.7×10^9 FLOPs per query. Approximate MIPS via:

FAISS HNSW. Hierarchical Navigable Small World graph: nodes connected by edges in a hierarchy of layers. Search traverses from the top layer (few long-range edges) to the bottom (many local edges), achieving $O(\log |\mathcal{C}|)$ query time with $> 95\%$ recall.

60.3 RAG-Fusion

RAG-Fusion (Rackauckas, 2023) improves retrieval by:

1. Generating m sub-queries from the original query using an LLM.
2. Retrieving top-k passages for each sub-query.
3. Re-ranking by Reciprocal Rank Fusion (RRF):

$$\text{RRF}(d) = \sum_{r=1}^{m} \frac{1}{k_0 + \text{rank}_r(d)}$$

where $\text{rank}_r(d)$ is the rank of document d in the r-th retrieval list and $k_0 = 60$ is a smoothing constant.

Example — RRF with 3 sub-queries

Document d^* ranks 1st, 3rd, 2nd across sub-queries. $k_0 = 60$.

$$\text{RRF}(d^*) = \frac{1}{61} + \frac{1}{63} + \frac{1}{62} \approx 0.01639 + 0.01587 + 0.01613 = 0.04839$$

A competing document ranking 2nd, 5th, 10th:

$$\frac{1}{62} + \frac{1}{65} + \frac{1}{70} = 0.01613 + 0.01538 + 0.01429 = 0.04580$$

d^* wins — consistent high relevance across diverse sub-queries is rewarded.

PART 61

Mechanistic Interpretability: Superposition and Circuits

Modern interpretability research studies how transformer weights store and compute information. We derive the superposition hypothesis mathematically, analyse sparse autoencoders, and discuss circuit discovery.

61.1 The Superposition Hypothesis

Superposition (Elhage et al. 2022)

A neural network represents n features in $d < n$ dimensions by storing them in *superposition*: as nearly orthogonal directions in $\mathbb{R}^d$. The cosine interference between features $\mathbf{f}_i$ and $\mathbf{f}_j$ is $|\mathbf{f}_i^\top \mathbf{f}_j|$. Superposition is viable when features are *sparse*: most features are near zero for most inputs.

Theoretical model. Consider n binary features with sparsity s (probability of being non-zero). Model weights $\mathbf{W} \in \mathbb{R}^{d \times n}$. Expected interference loss from feature i on feature j:

$$\mathbb{E}[\text{interference}_{ij}^2] = s^2 \cdot (\mathbf{w}_i^\top \mathbf{w}_j)^2$$

Total loss (with λ weighting robustness vs. compression):

$$\mathcal{L} = \sum_i (1 - \|\mathbf{w}_i\|^2)^2 + \lambda \sum_{i \neq j} (\mathbf{w}_i^\top \mathbf{w}_j)^2$$

Minimising: dense models ($d \geq n$) use orthogonal $\mathbf{w}_i$; compressed models ($d < n$) use near-uniform directions, accepting interference. As $s \to 0$ (sparser features), superposition becomes more efficient — more features can be stored.

61.2 Sparse Autoencoders (SAE)

To extract interpretable features from superposition, SAEs learn an overcomplete dictionary $\mathbf{D} \in \mathbb{R}^{d \times m}$ ($m \gg d$) by:

$$\mathbf{a} = \text{ReLU}(\mathbf{W}_e(\mathbf{x} - \mathbf{b}_{\text{pre}}) + \mathbf{b}_e), \qquad \hat{\mathbf{x}} = \mathbf{D}\mathbf{a} + \mathbf{b}_{\text{pre}}$$

$$\mathcal{L} = \|\mathbf{x} - \hat{\mathbf{x}}\|^2 + \lambda \|\mathbf{a}\|_1$$

The L_1 penalty enforces sparsity: each token activates only a small subset of the m dictionary features. Each active feature corresponds to an interpretable concept (e.g. "Python code", "sycophantic tone", "DNA sequences").

61.3 Circuit Discovery: Indirect Object Identification

Circuit (Wang et al. 2022)

A circuit is a subgraph of the full model — a small set of attention heads and MLPs — that implements a specific capability.

Example: Indirect Object Identification (IOI). For prompts like "When Mary and John went to the store, John gave a drink to _", the answer is "Mary" (the indirect object).

The IOI circuit in GPT-2 small (12 layers, 12 heads each) uses:

- **Duplicate Token Heads** (L0, L1): identify repeated names.
- **S-Inhibition Heads** (L8, L9): suppress the subject name (John) from the output.
- **Name Mover Heads** (L9, L10): move the indirect object token (Mary) to the output position.

Logit lens analysis. The *logit lens* projects intermediate residual stream states $\mathbf{h}_l$ through the unembedding matrix $\mathbf{W}_U \in \mathbb{R}^{V \times d}$:

$$\mathbf{p}_l = \text{softmax}(\mathbf{W}_U \mathbf{h}_l)$$

Tracking $\mathbf{p}_l$ across layers reveals how the prediction evolves from early layers (near-uniform) to late layers (peaked on the correct answer).

PART 62

RLHF and Constitutional AI

Reinforcement learning from human feedback formalises preference alignment as a reward learning problem. We derive the Bradley-Terry model, the PPO objective, and the Constitutional AI self-critique loop.

62.1 Bradley-Terry Preference Model

Given two responses y_1, y_2 to a prompt x, a human rater prefers y_1 with probability:

$$P(y_1 \succ y_2 | x) = \frac{e^{r(x,y_1)}}{e^{r(x,y_1)} + e^{r(x,y_2)}} = \sigma(r(x,y_1) - r(x,y_2))$$

where $r(x,y)$ is a latent reward model and σ is the sigmoid.

Reward model training. Given a dataset of comparisons $\{(x, y^+, y^-)\}$ (preferred y^+ vs. rejected y^-):

$$\mathcal{L}_{\text{RM}} = -\mathbb{E}_{(x,y^+,y^-)}\left[\log \sigma(r_\phi(x,y^+) - r_\phi(x,y^-))\right]$$

62.2 PPO for Language Models

The RLHF fine-tuning objective balances reward maximisation with KL penalty:

$$\max_{\pi_\theta} \; \mathbb{E}_{x\sim\mathcal{D},\, y\sim\pi_\theta(\cdot|x)}\left[r_\phi(x,y) - \beta\, \mathrm{KL}(\pi_\theta(\cdot|x) \| \pi_{\text{ref}}(\cdot|x))\right]$$

The optimal closed-form solution (without the constraint that π is a transformer):

$$\pi^*(y|x) = \frac{\pi_{\text{ref}}(y|x)\exp(r(x,y)/\beta)}{Z(x)}$$

DPO (Direct Preference Optimisation). Substituting π^* into the reward: $r(x,y) = \beta \log \frac{\pi(y|x)}{\pi_{\text{ref}}(y|x)} + \beta \log Z(x)$. Inserting into the Bradley-Terry loss eliminates $Z(x)$:

$$\mathcal{L}_{\text{DPO}} = -\mathbb{E}\left[\log \sigma\left(\beta \log \frac{\pi_\theta(y^+|x)}{\pi_{\text{ref}}(y^+|x)} - \beta \log \frac{\pi_\theta(y^-|x)}{\pi_{\text{ref}}(y^-|x)}\right)\right]$$

DPO directly trains the policy without a separate reward model — no PPO required.

62.3 Constitutional AI (CAI)

CAI (Bai et al. 2022) uses a set of principles (the "constitution") to generate self-critique and revision data:

1. **SL-CAI.** Model is prompted to critique its response according to a constitutional principle, then revise. Supervised fine-tuning on the revised responses.
2. **RL-CAI.** An AI Feedback (AIF) reward model is trained: the model generates preference data over its own responses using constitutional principles ("which response is more helpful and less harmful according to principle P?"). PPO (or DPO) then fine-tunes using this reward model.

Constitutional AI reduces dependence on human labellers for the harmful/harmless dimension by bootstrapping feedback from the model itself.

Part 63

Spiking Neural Networks and Quantum ML

We derive the Leaky Integrate-and-Fire neuron, its connection to rate coding, and introduce the mathematical foundations of quantum machine learning: qubits, Hilbert space, and variational circuits.

63.1 Leaky Integrate-and-Fire (LIF) Neuron

LIF Dynamics

Membrane potential $v(t)$ evolves as:

$$\tau_m \frac{dv}{dt} = -(v - v_{\text{rest}}) + R\,I(t)$$

When $v(t) \geq v_{\text{thresh}}$: emit spike, reset $v \leftarrow v_{\text{reset}}$.
Parameters: membrane time constant τ_m (typically 20ms), resistance R, resting potential v_{rest}, threshold v_{thresh}.

Rate coding. For constant input I, the steady-state firing rate is:

$$f = \left[\tau_m \ln \frac{R\,I - v_{\text{rest}} + v_{\text{reset}}}{R\,I - v_{\text{thresh}} + v_{\text{reset}}}\right]^{-1}$$

Surrogate gradient. During the forward pass, the spike function $S(v) = \Theta(v - v_{\text{thresh}})$ (Heaviside) is non-differentiable. In the backward pass, it is replaced by a differentiable surrogate:

$$S'(v) \approx \frac{1}{\pi} \cdot \frac{1}{1 + (\pi(v - v_{\text{thresh}}))^2}$$

(arctangent derivative), enabling backpropagation.

63.2 Qubit Hilbert Space

A qubit lives in $\mathbb{C}^2$. State: $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, $|\alpha|^2 + |\beta|^2 = 1$.

Bloch Sphere Representation

Any pure qubit state can be written as:

$$|\psi\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\phi}\sin\frac{\theta}{2}|1\rangle$$

where $\theta \in [0, \pi]$, $\phi \in [0, 2\pi)$. This is a point on the unit sphere S^2.

Measurement. Measuring in the computational basis collapses $|\psi\rangle$ to $|0\rangle$ with probability $|\alpha|^2$ and $|1\rangle$ with probability $|\beta|^2$.

Entanglement. Two-qubit state $|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ cannot be written as $|\psi_1\rangle \otimes |\psi_2\rangle$ — this is an entangled (Bell) state.

63.3 Variational Quantum Circuits (VQC)

A VQC applies a sequence of parameterised unitary gates $U(\square)$ to an initial state $|0\rangle^{\otimes n}$, then measures an observable $\mathcal{O}$:

$$f(\square) = \langle 0|^{\otimes n} U(\square)^\dagger \, \mathcal{O} \, U(\square)|0\rangle^{\otimes n}$$

Parameter shift rule (gradient computation):

$$\frac{\partial f}{\partial \theta_k} = \frac{1}{2}\left[f\left(\theta_k + \frac{\pi}{2}\right) - f\left(\theta_k - \frac{\pi}{2}\right)\right]$$

This is an exact gradient (not finite difference) for gates of the form $e^{-i\theta G}$ where $G^2 = \mathbf{I}$. VQCs are trained by classical optimisers using these exact quantum gradients.

Part 64

Neural Architecture Search: DARTS and Efficient NAS

Differentiable Architecture Search (DARTS) relaxes the discrete architecture space to a continuous one, enabling gradient-based optimisation of the architecture jointly with the weights.

64.1 The NAS Problem

Given a search space $\mathcal{A}$ of architectures, find:

$$a^* = \arg\min_{a \in \mathcal{A}} \; \mathcal{L}_{\text{val}}(w^*(a), a)$$

where $w^*(a) = \arg\min_w \mathcal{L}_{\text{train}}(w, a)$. This is a bilevel optimisation problem.

Reinforcement learning NAS (Zoph & Le, 2017) samples architectures from a controller policy and optimises by REINFORCE: expensive ($\sim 10^4$ GPU-days).

64.2 DARTS: Continuous Relaxation

Replace the discrete choice of operation $o \in \mathcal{O}$ at each edge (i, j) with a softmax mixture:

$$\bar{o}^{(i,j)}(\mathbf{x}) = \sum_{o \in \mathcal{O}} \frac{e^{\alpha_o^{(i,j)}}}{\sum_{o'} e^{\alpha_{o'}^{(i,j)}}} \cdot o(\mathbf{x})$$

where $\square \in \mathbb{R}^{|\mathcal{O}|}$ are architecture parameters learned by gradient descent.

Bilevel objective. Joint optimisation:

$$\begin{aligned} &\min_{\square} \; \mathcal{L}_{\text{val}}(w^*(\square), \square) \\ &\text{s.t.} \quad w^*(\square) = \arg\min_w \mathcal{L}_{\text{train}}(w, \square) \end{aligned}$$

DARTS approximates: one gradient step on w (training loss), one gradient step on $\square$ (validation loss), alternating. The architecture gradient includes a second-order term:

$$\nabla_{\square} \mathcal{L}_{\text{val}} \approx \nabla_{\square} \mathcal{L}_{\text{val}}(w, \square) - \xi \, \nabla^2_{\square, w} \mathcal{L}_{\text{train}} \cdot \nabla_w^{-1} \mathcal{L}_{\text{train}} \cdot \nabla_w \mathcal{L}_{\text{val}}$$

The second-order term is computed via finite differences to avoid storing the Hessian.

64.3 Deriving the Final Architecture

After training, the discrete architecture is obtained by argmax over operations at each edge:

$$o^{(i,j)} = \arg\max_{o \neq \text{zero}} \; \alpha_o^{(i,j)}$$

DARTS instability. DARTS is known to favour skip connections (identity, zero) over meaningful operations because they have near-zero training loss but do not contribute to expressivity. Regularisation methods (GDAS, SNAS) address this by stochastic operation sampling.

Part 65

Autoregressive Models: MADE, PixelCNN, VQ-VAE and Summary of the Series

We derive the autoregressive factorisation, masked autoencoders, PixelCNN's causal convolutions, and VQ-VAE's vector quantisation. The part closes with a synthesis across all five volumes.

65.1 Autoregressive Factorisation

The chain rule of probability:

$$p(\mathbf{x}) = p(x_1) \prod_{i=2}^{n} p(x_i | x_1, \dots, x_{i-1})$$

Any joint distribution can be factored this way. The challenge is parameterising $p(x_i | x_{<i})$ for $i = 1, \dots, n$ with a neural network.

65.2 MADE: Masked Autoencoder for Distribution Estimation

MADE (Germain et al. 2015) computes all conditionals $p(x_i | x_{<i})$ in a single forward pass via *masking*:

MADE Connectivity Constraint

Assign each hidden unit h an order $m(h) \in \{1, \dots, d-1\}$ and each input/output unit an order $m(x_i) = i$.
Mask weight $M_{kl} = \mathbf{1}[m(k) \geq m(l)]$. This ensures output $\hat{x}_i$ depends only on x_j for $j < i$.

Key property. For d inputs and hidden sizes $\mathbf{s}$, MADE masks the weight matrices:

$$\hat{\mathbf{x}} = \text{softmax}\Big(\mathbf{M}_{W_2} \odot \mathbf{W}_2 \cdot \sigma\Big((\mathbf{M}_{W_1} \odot \mathbf{W}_1)\mathbf{x}\Big)\Big)$$

A single forward pass computes all d conditionals simultaneously.

65.3 PixelCNN: Causal Convolutions

PixelCNN (van den Oord et al. 2016) models images autoregressively: pixel $x_{i,j}$ depends on all pixels above row i and to the left in row i.

Masked convolutions. A convolutional filter is zeroed in its lower-left portion:

$$K_{\text{mask A}} = K \odot \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \qquad K_{\text{mask B}} = K \odot \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Mask A (first layer): excludes the current pixel. Mask B (subsequent layers): includes the current pixel's previous-layer representation.

Blind spot problem. Standard masked convolutions have a "blind spot" for pixels directly above the current position. The Gated PixelCNN resolves this by separating vertical (column scan) and horizontal (row scan) stacks.

65.4 VQ-VAE: Vector Quantised Variational Autoencoder

VQ-VAE learns a discrete latent space via a learnable codebook $\mathbf{E} = \{\mathbf{e}_k\}_{k=1}^{K} \subset \mathbb{R}^{d_z}$.

VQ-VAE Quantisation

Encoder output: $\mathbf{z}_e = E(\mathbf{x})$. Quantised latent:

$$\mathbf{z}_q = \mathbf{e}_{k^*}, \qquad k^* = \arg\min_k \|\mathbf{z}_e - \mathbf{e}_k\|^2$$

Decoder input: $\mathbf{z}_q$ (passed as if it were a continuous vector).

Straight-through estimator. The argmin is not differentiable. During backpropagation, the gradient $\dfrac{\partial \mathcal{L}}{\partial \mathbf{z}_q}$ is copied directly to $\mathbf{z}_e$ (straight-through):

$$\frac{\partial \mathcal{L}}{\partial \mathbf{z}_e} \approx \frac{\partial \mathcal{L}}{\partial \mathbf{z}_q}$$

Loss.

$$\mathcal{L} = \underbrace{\|\mathbf{x} - D(\mathbf{z}_q)\|^2}_{\text{reconstruction}} + \underbrace{\|\mathrm{sg}[\mathbf{z}_e] - \mathbf{e}_{k^*}\|^2}_{\text{codebook update}} + \underbrace{\beta \|\mathbf{z}_e - \mathrm{sg}[\mathbf{e}_{k^*}]\|^2}_{\text{commitment}}$$

where $\mathrm{sg}[\cdot]$ denotes stop-gradient. The codebook term moves $\mathbf{e}_{k^*}$ toward $\mathbf{z}_e$; the commitment term prevents $\mathbf{z}_e$ from growing unboundedly.

Example — VQ-VAE codebook update, $K = 4$, $d_z = 2$

$\mathbf{z}_e = (1.2, 0.8)$. Codebook: $\mathbf{e}_1 = (0,0)$, $\mathbf{e}_2 = (1,1)$, $\mathbf{e}_3 = (2,0)$, $\mathbf{e}_4 = (0,2)$.

$$\|\mathbf{z}_e - \mathbf{e}_1\|^2 = 1.44 + 0.64 = 2.08, \quad \|\mathbf{z}_e - \mathbf{e}_2\|^2 = 0.04 + 0.04 = 0.08$$

$$\|\mathbf{z}_e - \mathbf{e}_3\|^2 = 0.64 + 0.64 = 1.28, \quad \|\mathbf{z}_e - \mathbf{e}_4\|^2 = 1.44 + 1.44 = 2.88$$

$k^* = 2$. Codebook update step ($\eta = 0.1$): $\mathbf{e}_2 \leftarrow (1,1) + 0.1 \cdot ((1.2, 0.8) - (1,1)) = (1.02, 0.98)$.

65.5 Synthesis: The Five-Volume Arc

Vol.	Theme	Mathematical Core
I	Foundations	Backpropagation chain rule, MLP universality, convolutional weight-sharing, attention quadratic complexity.
II	Probabilistic & Generative	ELBO variational bound, reparameterisation gradient, Wasserstein distance, score matching identity.
III	Geometric & Graph	Laplacian eigendecomposition, group equivariance, Monge-Kantorovich duality, $\mathrm{SE}(3)$ frames.
IV	Optimisation & Robustness	Fisher-Rao metric, PAC-Bayes KL bound, adversarial min-max, adjoint ODE, Nyquist sampling.
V	Scale & Emergence	Chinchilla power laws, superposition in $\mathbb{R}^d$, linear attention = SSM, DPO preference optimisation.

The invariants of the series. Every derivation proceeds step by step with no "it can be shown." Every symbol is defined at first use. Every abstract result is grounded in a worked numerical example. Every geometric interpretation is made explicit. These four commitments hold from Part 0 (scalar derivatives) through Part 65 (quantum variational circuits) — a continuous thread from first principles to the research frontier.

PART 53 — SUPPLEMENT

Linear Complexity Attention: Performer and Hyper-Attention

We derive the FAVOR+ approximation underlying the Performer architecture, analyse the Random Feature Map construction for kernel approximation, and survey Hyper-Attention's hash-based exact sparsification.

53.5 Random Feature Maps and FAVOR+

The Performer (Choromanski et al. 2021) approximates the softmax kernel $k(q,k) = \exp(q^\top k/\sqrt{d})$ via random feature maps, enabling $O(nd)$ attention instead of $O(n^2)$.

Positive random features for softmax. For $q, k \in \mathbb{R}^d$, write:

$$\exp\left(\frac{q^\top k}{\sqrt{d}}\right) = \exp\left(-\frac{\|q\|^2}{2\sqrt{d}}\right)\exp\left(-\frac{\|k\|^2}{2\sqrt{d}}\right)\exp\left(\frac{(q+k)^\top(q+k)}{2\sqrt{d}} - \frac{\|q\|^2+\|k\|^2}{2\sqrt{d}}\right)$$

The standard approach uses the kernel identity for the Gaussian kernel $\exp(-\|q-k\|^2/2)$, which can be approximated via random Fourier features (Rahimi & Recht, 2007):

$$\exp(-\|q-k\|^2/2) \approx \phi(q)^\top\phi(k), \qquad \phi(x) = \frac{1}{\sqrt{m}}(\cos(\omega_1^\top x + b_1), \dots, \cos(\omega_m^\top x + b_m))$$

with $\omega_j \sim \mathcal{N}(0, \mathbf{I}_d)$, $b_j \sim \text{Uniform}[0, 2\pi]$.

However, for the softmax kernel the Fourier features can be negative, leading to numerical instability. FAVOR+ (Fast Attention Via positive Orthogonal Random features) uses *positive* features:

$$\phi^+(x) = \frac{e^{-\|x\|^2/2}}{\sqrt{m}}(e^{\omega_1^\top x}, \dots, e^{\omega_m^\top x}), \qquad \omega_j \sim \mathcal{N}(0, \mathbf{I}_d)$$

Unbiasedness. $\mathbb{E}_\omega[e^{\omega^\top q} \cdot e^{\omega^\top k}] = e^{(q^\top k + \|q\|^2/2 + \|k\|^2/2)}$. Multiplying by $e^{-(\|q\|^2+\|k\|^2)/2}$ recovers $\exp(q^\top k)$. So:

$$\mathbb{E}[\phi^+(q)^\top\phi^+(k)] = \exp(q^\top k)$$

The approximation is unbiased. Orthogonal random features reduce variance: rather than drawing ω_j i.i.d., they are sampled as random orthogonal matrices, keeping all directions well-separated and reducing the approximation error by a constant factor.

Linear attention via FAVOR+. Replacing the softmax with the FAVOR+ kernel approximation in attention:

$$\text{Attn}(q_i, K, V) \approx \frac{\phi^+(q_i)^\top \sum_j \phi^+(k_j) v_j^\top}{\phi^+(q_i)^\top \sum_j \phi^+(k_j)}$$

Define $\mathbf{S} = \sum_j \phi^+(k_j) v_j^\top \in \mathbb{R}^{m \times d_v}$ and $\mathbf{z} = \sum_j \phi^+(k_j) \in \mathbb{R}^m$. These can be computed once in $O(nmd)$; then each query costs $O(md_v)$. Total: $O(nmd)$ with $m \ll n$ — linear in n.

EXAMPLE — FAVOR+ complexity, $n = 65536$, $d = 64$, $m = 256$

Standard attention: $65536^2 \times 64 = 2.7 \times 10^{11}$ FLOPs, 16 GB for the attention matrix.
FAVOR+: $65536 \times 256 \times 64 \times 2 = 2.1 \times 10^9$ FLOPs. Speedup: $\approx 130\times$.
Approximation error (Monte Carlo): $\mathbb{E}[|\phi^+(q)^\top\phi^+(k) - e^{q^\top k}|^2] \le \frac{e^{2\|q\|\|k\|}}{m}$. For $\|q\| = \|k\| = 2$ and $m = 256$: error bound $\le e^8/256 \approx 11.5$. Error scales as $1/\sqrt{m}$; increasing

to $m = 4096$ reduces it by $4\times$.

53.6 Hyper-Attention

Hyper-Attention (Han et al. 2023) achieves $O(n\sqrt{n})$ *exact* sparse attention by identifying, for each query q_i, the keys k_j with large $q_i^\top k_j$ without computing all n^2 inner products.

Locality-Sensitive Hashing (LSH) for MIPS. For maximum inner-product search, the signed random projection:

$$h(x) = \text{sign}(\omega^\top x), \qquad \omega \sim \mathcal{N}(0, \mathbf{I}_d)$$

satisfies $P[h(q) = h(k)] = 1 - \frac{\arccos(q^\top k/(\|q\|\|k\|))}{\pi}$. Queries and keys with large inner products (small angle) hash to the same bucket with high probability.

Algorithm. For each head:

1. Normalise keys: $\hat{k}_j = k_j/\|k_j\|$.
2. Apply r hash functions to bucket $\hat{K}$ into groups of size $\approx B$.
3. For each query q_i, find its hash bucket (cost $O(r)$); attend only within the bucket (cost $O(B \cdot d)$).

With bucket size $B = O(\sqrt{n})$, total cost $O(n\sqrt{n}d)$. Collision probability analysis shows: keys that are "heavy hitters" (large $q_i^\top k_j$) fall in the same bucket with probability $\geq 1 - \delta$ for $r = O(\log(n/\delta))$ hash functions.

PART 54 — SUPPLEMENT

Positional Encoding: T5 Relative Bias, Alibi Generalisation and xPos

We derive the T5 relative attention bias, analyse how ALiBi relates to exponential decay kernels, and prove the xPos stability theorem for long-range extrapolation.

54.4 T5 Relative Position Bias

T5 (Raffel et al. 2020) uses a learned scalar bias b_{ij} added to attention logits, depending only on the relative position bucket $r(i-j)$:

$$a_{ij} = \frac{q_i^\top k_j}{\sqrt{d_k}} + b_{r(i-j)}$$

The bucket function clips and logarithmically bins large distances:

$$r(\delta) = \begin{cases} \delta & |\delta| \le K_0 \\ K_0 + \lfloor \log_s(|\delta|/K_0) \cdot (K_1 - K_0) \rfloor & |\delta| > K_0 \end{cases}$$

with $K_0 = 16$, $K_1 = 32$, $s = \exp(\log(L_{\max}/K_0)/(K_1 - K_0))$.

Rationale. At short distances (within K_0 tokens), exact position matters; at long distances, only logarithmic resolution is needed. The total number of learned bias values is $K_1 = 32$ per head, independent of context length. T5 therefore generalises to longer sequences without architectural changes.

Comparison to RoPE. RoPE encodes absolute positions as rotations but achieves relative attention through the dot-product identity. T5 relative bias directly models position as a bias. The fundamental difference: RoPE's positional signal is *multiplicative* (modifies the query-key dot product via rotation), while T5's is *additive* (shifts logits). Additive biases interact poorly with attention entropy: if b_{ij} is large, it can dominate the learned similarity signal.

54.5 ALiBi as an Exponential Decay Kernel

Recall ALiBi: $a_{ij} = q_i^\top k_j/\sqrt{d_k} - m_h|i-j|$ with head-specific slope m_h.

ALiBi as a Kernel

ALiBi attention computes:

$$\text{Attn}_h(i) = \sum_j \text{softmax}\left(\frac{q_i^\top k_j}{\sqrt{d_k}} - m_h|i-j|\right)_j v_j$$

The term $-m_h|i-j|$ is the log of an exponential decay kernel: $e^{-m_h|i-j|}$. So ALiBi implements *locally-weighted attention* with an exponential window.

Effective receptive field. At slope m_h, the attention weight for a token at distance δ decays as $e^{-m_h\delta}$ relative to the same-position token. The effective context window (where attention weight $\ge 1/e$ of the local maximum) is $\delta^* = 1/m_h$. For $H = 8$ heads with slopes $\{2^{-1}, 2^{-2}, \ldots, 2^{-8}\}$: windows $\{2, 4, 8, 16, 32, 64, 128, 256\}$ tokens. The ensemble covers multi-scale context.

Length extrapolation analysis. At inference with $L_{\text{test}} > L_{\text{train}}$: new positions $i > L_{\text{train}}$ attend to tokens at distance $\delta \leq i < L_{\text{test}}$. ALiBi's penalty $-m_h\delta$ grows without bound, so distant tokens have $e^{-m_h L_{\text{test}}} \approx 0$ weight — the model effectively ignores them. This graceful degradation (not explosion) is why ALiBi extrapolates: the model's predictions change only because it now has more local context, not because unfamiliar absolute positions destabilise the computation.

54.6 xPos: Extended Positional Embeddings

xPos (Sun et al. 2023) modifies RoPE to improve both short-range and long-range stability by introducing a *decay factor* γ_i per dimension pair:

$$\tilde{q}_{2i} = q_{2i}\gamma_i^{m/2}, \quad \tilde{k}_{2i} = k_{2i}\gamma_i^{-m/2}$$

with $\gamma_i = \exp(-\delta\theta_i^{2/d})$ for a small $\delta > 0$.

The key-query dot product becomes:

$$(\mathbf{R}_m^{(\gamma)}\mathbf{q})^\top(\mathbf{R}_n^{(\gamma)}\mathbf{k}) = \sum_i \gamma_i^{(m-n)/2}[q_{2i}k_{2i}\cos(m-n)\theta_i + q_{2i+1}k_{2i+1}\cos(m-n)\theta_i + \ldots]$$

The factor $\gamma_i^{(m-n)/2}$ decays as distance $|m-n|$ grows, creating an inductive bias towards local context. For high-frequency dimensions (θ_i large): $\gamma_i \approx 1$ (decay negligible — they're already local). For low-frequency dimensions (θ_i small): $\gamma_i < 1$ (decay applied — prevents spurious long-range correlations at distances beyond training).

Example — xPos decay factor, $d = 128$, $\delta = 0.4$, dimension $i = 32$

$\theta_{32} = 10000^{-64/128} = 10000^{-1/2} = 0.01$. $\gamma_{32} = \exp(-0.4 \times 0.01^{2/128}) = \exp(-0.4 \times 10000^{-1/64})$.
$10000^{1/64} = e^{\ln(10000)/64} = e^{0.144} \approx 1.155$. So $\gamma_{32} \approx \exp(-0.4/1.155) \approx e^{-0.346} \approx 0.707$. At distance $|m-n| = 100$: decay factor $0.707^{50} \approx 10^{-8}$. This dimension's contribution to the dot product is suppressed at long range, preventing it from creating spurious associations between tokens 100 apart.

PART 55 — SUPPLEMENT

State Space Models: Diagonal Structures, S5, and Mamba-2

We prove the HiPPO optimality theorem, derive the S5 multi-input multi-output generalisation, and analyse Mamba-2's SSD (State Space Duality) layer which unifies SSMs and chunked linear attention.

55.4 HiPPO Optimality Theorem

HiPPO-LegT (Legendre Time)

Let ω_t be the uniform measure on $[t-\theta, t]$ (sliding window of length θ). The optimal polynomial projection is:

$$c_n(t) = \int_{t-\theta}^{t} u(s) \frac{(2n+1)^{1/2}}{\theta^{1/2}} P_n\left(\frac{2(s-t)}{\theta} + 1\right) ds$$

where P_n is the n-th Legendre polynomial on $[-1, 1]$. The ODE for $\mathbf{c}(t)$ is exactly $\dot{\mathbf{c}} = \mathbf{A}\mathbf{c} + \mathbf{B}u$ with the triangular HiPPO matrix.

Proof sketch. Define the approximation error $\mathcal{E}(t) = \int \|u(s) - \hat{u}(s,t)\|^2 \omega_t(ds)$ where $\hat{u}(s,t) = \sum_{n=0}^{N-1} c_n(t) P_n^{(t)}(s)$ is the running polynomial approximation. The ODE $\dot{\mathbf{c}} = \mathbf{A}\mathbf{c} + \mathbf{B}u$ follows from differentiating the inner product $c_n(t) = \langle u, P_n^{(t)} \rangle_{\omega_t}$ with respect to t. The Leibniz rule gives contributions from (a) the change in u's values (the $\mathbf{B}u$ term) and (b) the change in the window ω_t (the $\mathbf{A}\mathbf{c}$ term, which accounts for the window sliding forward). The HiPPO matrix $\mathbf{A}$ is the unique matrix ensuring $\frac{d}{dt}\mathcal{E}(t) \leq 0$ for all u, i.e. the approximation never degrades due to the state evolution alone.

Memory capacity. With N state dimensions, HiPPO-LegT can represent the last θ time steps of input with polynomial approximation error $O(\theta^{N+1}/N!)$ (standard Legendre approximation theory). For $N = 256$ and $\theta = 1000$: error $\sim (1000/256)^{256}/256! \approx 0$ — effectively perfect memory for smooth inputs.

55.5 S5: Multi-Input Multi-Output SSM

S5 (Smith et al. 2023) replaces the scalar-to-scalar SSM of S4 with a MIMO (multiple input, multiple output) formulation acting on vector-valued sequences:

$$\mathbf{h}_k = \bar{\mathbf{A}}\mathbf{h}_{k-1} + \bar{\mathbf{B}}\mathbf{u}_k \in \mathbb{R}^N, \qquad \mathbf{y}_k = \mathbf{C}\mathbf{h}_k + \mathbf{D}\mathbf{u}_k \in \mathbb{R}^H$$

with $\mathbf{u}_k \in \mathbb{R}^H$ (full model dimension), $\bar{\mathbf{A}} \in \mathbb{R}^{N \times N}$ diagonal in the complex basis.

Parallelism. S5 computes all L outputs in $O(L \log L)$ using the parallel prefix scan (as in Mamba) rather than convolution, making it more flexible for variable-length sequences. The diagonal $\bar{\mathbf{A}}$ means the scan over N state dimensions decomposes into N independent scalar recurrences, each parallelisable.

Initialisation from HiPPO. The N diagonal entries of $\bar{\mathbf{A}}$ are initialised to the eigenvalues of the HiPPO matrix $\mathbf{A}$, providing theoretically principled long-range memory at initialisation.

55.6 Mamba-2 and State Space Duality (SSD)

Mamba-2 (Dao & Gu, 2024) introduces the SSD framework, proving a formal equivalence between certain structured SSMs and chunked linear attention.

SSD Layer

For input sequence $(\mathbf{x}_1, \ldots, \mathbf{x}_L)$, the SSD layer computes:

$$\mathbf{y}_t = \sum_{s \le t} \underbrace{\left(\prod_{r=s+1}^{t} a_r \right)}_{=: \mathbf{L}_{ts}} \mathbf{B}_s^\top \mathbf{x}_s \cdot \mathbf{C}_t$$

where $a_t \in \mathbb{R}$ (scalar decay, input-dependent), $\mathbf{B}_t \in \mathbb{R}^{N \times d}$, $\mathbf{C}_t \in \mathbb{R}^{d \times N}$.

Matrix form. Define the $L \times L$ semiseparable matrix $\mathbf{M}$ with $M_{ts} = \mathbf{L}_{ts}$ for $s \le t$ and 0 otherwise. Then $\mathbf{y} = \mathbf{M}(\mathbf{B}\mathbf{x})\mathbf{C}^\top$. This is exactly linear attention with a structured mask: $\mathbf{M}$ replaces the causal mask $\mathbf{1}[s \le t]$ with the exponential decay $\prod_{r=s+1}^{t} a_r$.

Chunked computation. Partition the sequence into chunks of size C. Within each chunk: compute the full $C \times C$ SSD matrix (quadratic in C, not L). Across chunks: pass the hidden state recurrently (linear in L/C). Total cost: $O(LC + L/C \cdot N^2)$. Optimal chunk size $C = N$ gives $O(LN)$.

Example — SSD vs Mamba-1 comparison, $L = 8192$, $N = 64$, $d = 2048$

Mamba-1 (selective scan): $L \times N \times d = 8192 \times 64 \times 2048 = 1.07 \times 10^9$ FLOPs (sequential scan).
SSD chunked ($C = 64$): $L/C \times C^2 \times d + L/C \times N^2 \times d = 128 \times 4096 \times 2048 + 128 \times 4096 \times 2048 = 2 \times 128 \times 4096 \times 2048 \approx 2.1 \times 10^9$ FLOPs.
SSD is $2\times$ more FLOPs but achieves $3\times$ higher hardware utilisation on H100 due to tensor core-friendly matrix multiplications (vs. scan's sequential dependencies), resulting in $1.5\times$ faster wall-clock time.

PART 57 — SUPPLEMENT

Scaling Laws: Inference Compute, Data Mixtures and the Compute-Optimal Frontier

We analyse inference-time compute scaling, derive the optimal data mixture from a generalised Chinchilla model, and discuss the compute-optimal Pareto frontier in the (N, D, C) space.

57.4 Inference-Time Compute Scaling

Snell et al. (2024) show that inference compute C_{inf} also follows scaling laws. For chain-of-thought length T on problems requiring depth k:

$$L(T) \approx L_\infty + \frac{A}{T^\alpha}$$

where L_∞ is the irreducible error and $\alpha \approx 0.5$. Doubling inference tokens reduces error by $\approx 30\%$ (consistent with $\alpha = 0.5$).

Training vs inference compute tradeoff. For a fixed total compute $C_{\text{total}} = C_{\text{train}} + n_{\text{queries}} \times C_{\text{inf}}$, minimise loss by allocating:

$$\frac{C^*_{\text{inf}}}{C^*_{\text{train}}} \propto \frac{\beta_{\text{inf}}/\alpha_{\text{inf}}}{\beta_{\text{train}}/\alpha_{\text{train}}}$$

where β, α are the exponents in the respective scaling laws. Current estimates suggest that for reasoning tasks, inference compute is underutilised relative to training: the optimal ratio may be $C_{\text{inf}} \approx 0.1 C_{\text{train}}$ per query for long-lived deployed models.

57.5 Data Mixture Optimisation

Let the training corpus consist of K domains with token counts $D_1, \ldots, D_K$ and mixture weights $\lambda_k = D_k / \sum D_k$. The loss on domain k follows:

$$L_k(N, D_k) = E_k + \frac{A_k}{N^{\alpha_k}} + \frac{B_k}{D_k^{\beta_k}}$$

with domain-specific parameters $(E_k, A_k, \alpha_k, B_k, \beta_k)$. The combined evaluation loss (on a target mixture μ_k) is:

$$L_{\text{eval}} = \sum_k \mu_k L_k(N, D_k)$$

Optimal mixture for Chinchilla allocation. Under the constraint $\sum D_k = D^*$ (Chinchilla-optimal total tokens), the per-domain allocation solves:

$$\min_{\{D_k\}} \sum_k \mu_k L_k(N, D_k) \quad \text{s.t.} \quad \sum_k D_k = D^*$$

By Lagrange multipliers: at the optimum,

$$\mu_k \frac{dL_k}{dD_k} = \mu_j \frac{dL_j}{dD_j} \quad \forall k, j$$

$$\Rightarrow \quad \mu_k B_k \beta_k D_k^{-\beta_k - 1} = \text{const} \quad \Rightarrow \quad D_k^* \propto (\mu_k B_k \beta_k)^{1/(\beta_k + 1)}$$

Example — Optimal mixture for code and web text

Two domains: code ($k=1$, $B_1=1.5$, $\beta_1=0.30$, $\mu_1=0.2$) and web ($k=2$, $B_2=1.0$, $\beta_2=0.25$, $\mu_2=0.8$).
$D_1^* \propto (0.2\times 1.5\times 0.30)^{1/1.30} = (0.09)^{0.769} \approx 0.126$.
$D_2^* \propto (0.8\times 1.0\times 0.25)^{1/1.25} = (0.20)^{0.800} \approx 0.247$.
Normalised: $\lambda_1^* = 0.126/(0.126+0.247) = 0.34$, $\lambda_2^* = 0.66$.
Optimal code fraction is 34% (vs. the target evaluation weight of 20%) — code is up-weighted because it has a favourable scaling exponent ($\beta_1 > \beta_2$).

57.6 The Compute-Optimal Pareto Frontier

The Chinchilla result identifies the optimal (N^*, D^*) for a given C, but this is a single point on a broader Pareto frontier in the (N, D) plane. Define:

- **Train-optimal**: minimise $L(N, D)$ subject to $6ND = C$ (Chinchilla).
- **Deploy-optimal**: minimise inference cost $\propto N$ subject to $L(N,D) \le L^*$ and $6ND \le C_{\text{budget}}$. This favours smaller N and larger D.
- **Quality-optimal**: minimise L subject to $N \le N_{\max}$ (hardware constraint). This favours using all available N and maximising D.

The under-trained large model regime. Llama (Touvron et al. 2023) deliberately trains a 7B model on 1T tokens (vs. Chinchilla-optimal $\sim 140B$ tokens for 7B), trading training efficiency for inference efficiency. The resulting model achieves lower loss than a Chinchilla-optimal 7B (which would train on $\sim 140B$ tokens) because $L(7B, 10^{12}) < L(7B, 1.4\times 10^{11})$: more data is always better given fixed model size.

The insight: Chinchilla minimises loss per FLOP of training compute. If the model will be queried billions of times, the relevant metric is loss per FLOP of total (training + inference) compute. In this regime, smaller models trained much longer dominate.

PART 58 — SUPPLEMENT

Grokking and Delayed Generalisation: Weight Norms, Circuits and Phase Transitions

We derive the weight norm theory of grokking, formalise the circuit formation hypothesis, and analyse the sharp phase transitions in modular arithmetic tasks as a case study in mechanistic interpretability at scale.

58.4 The Grokking Phenomenon

Grokking (Power et al. 2022): a model first memorises training data (near-zero training loss), then after many additional gradient steps, suddenly generalises (validation loss drops sharply).

Weight norm theory. Neel Nanda et al. (2023) show that grokking correlates with the norm of the weight matrices. Define $R = \|W_{\text{memo}}\|/\|W_{\text{gen}}\|$ as the ratio of memorisation circuit weight norm to generalisation circuit weight norm. Early in training, $R > 1$: the memorisation circuit dominates. Grokking occurs when $R < 1$: weight decay (L2 regularisation) penalises the larger memorisation circuit more than the smaller, efficient generalisation circuit, until the memorisation circuit is sufficiently suppressed.

Formalisation. Let $W = W_{\text{memo}} + W_{\text{gen}}$ (decomposition into circuits). With weight decay λ, the training loss evolves as:

$$\frac{d}{dt}\|W\|^2 = -2\lambda\|W\|^2 - 2\eta\nabla_W \mathcal{L}_{\text{train}} \cdot W$$

The generalisation circuit develops a smaller norm (efficient algorithm), so weight decay eventually kills the memorisation circuit. The transition is sharp because the generalisation circuit's performance is near-chance until it reaches sufficient magnitude (non-linearly gated by softmax and layer norms).

Grokking Threshold (informal)

Let $w_g = \|W_{\text{gen}}\|$. The validation loss drops sharply when $w_g > w_g^*$ where w_g^* is a threshold set by the task complexity. For modular addition mod p: $w_g^* \approx O(\sqrt{p})$ (scales with the number of frequencies needed to represent the discrete Fourier transform solution).

58.5 Modular Arithmetic as a Mechanistic Case Study

For the task $a + b \pmod{p}$, the trained transformer learns a *Fourier algorithm*:

$$\text{output}(a, b) = \sum_{\omega \in \mathcal{F}} \alpha_\omega \cos\left(\frac{2\pi\omega(a+b)}{p}\right)$$

where $\mathcal{F} \subset \{0, \ldots, p/2\}$ is a sparse set of frequencies.

Why Fourier? The modular addition structure implies that the natural basis for functions $f : \mathbb{Z}_p \times \mathbb{Z}_p \to \mathbb{Z}_p$ is Fourier. The group $\mathbb{Z}_p$ has a natural discrete Fourier transform:

$$\hat{f}(\omega) = \sum_{a=0}^{p-1} f(a) e^{-2\pi i \omega a/p}$$

Functions that are invariant under shifts of $a + b$ are exactly the functions whose Fourier support is on frequencies ω with $\hat{f}(\omega) = \hat{f}(\omega)e^{2\pi i\omega/p}$ — i.e. the full Fourier basis. The transformer selects a sparse subset because L_1 regularisation (implicit from gradient noise) promotes sparsity in the frequency domain.

Embedding analysis. The trained token embeddings $W_E \in \mathbb{R}^{p\times d}$ have row a lying approximately on a 2D subspace spanned by $(\cos(2\pi\omega a/p), \sin(2\pi\omega a/p))$ for the dominant frequency ω. This is directly measurable by computing the PCA of W_E and checking alignment with the Fourier basis vectors.

PART 59 — SUPPLEMENT

Tree-of-Thought, Self-Consistency and Search-Augmented Reasoning

We formalise Tree-of-Thought as a search problem in a reasoning tree, derive the majority-vote self-consistency estimator, and analyse Monte Carlo Tree Search (MCTS) applied to language model reasoning.

59.4 Tree-of-Thought (ToT)

ToT (Yao et al. 2023) models reasoning as a tree search where each node is a partial reasoning path and edges are next-thought generation steps.

ToT as a Markov Decision Process

State: $s_t = (x, z_{1:t})$ — original problem x and partial reasoning $z_{1:t}$. Action: next thought $z_{t+1} \sim p_\theta(\cdot|s_t)$. Reward: $V(s_T) \in [0,1]$ — a value function (another LLM call) estimating solution quality.

BFS-ToT. At each step, maintain a beam of width b. Generate k candidate thoughts for each beam element; score with V; keep top b. Total LLM calls: $T \times b \times (k+1)$.

DFS-ToT. Depth-first with pruning: if $V(s_t) < \tau$, prune and backtrack. Efficient for tasks with clear intermediate failures (e.g. constraint satisfaction).

Complexity advantage of ToT over CoT. Standard CoT generates a single path of length T: $O(T)$ LLM calls. ToT with BFS width b generates $O(Tb)$ calls but explores a much larger reasoning space. The effective branching factor is b, enabling the system to solve problems that are hard for greedy left-to-right reasoning but easy for a b-way branching search.

Formally, if the problem has a reasoning tree of depth T and branching factor b^*, and the LLM's per-step accuracy is p, the probability of reaching the solution is:

$$P_{\text{CoT}} = p^T \qquad \text{vs.} \qquad P_{\text{ToT}} = 1 - (1 - p^T)^b$$

For $T = 4$, $p = 0.7$, $b = 5$: $P_{\text{CoT}} = 0.7^4 = 0.24$, $P_{\text{ToT}} = 1 - (1 - 0.24)^5 \approx 0.93$.

59.5 Self-Consistency as Majority Voting

Wang et al. (2022) sample m independent chain-of-thought paths $z^{(1)}, \ldots, z^{(m)}$ and take the majority answer:

$$\hat{y} = \arg\max_y \sum_{i=1}^{m} \mathbf{1}[y^{(i)} = y]$$

Error analysis. Assume each path gives the correct answer independently with probability $p > 1/2$. The majority vote errs when more than $m/2$ paths are wrong:

$$P_{\text{error}}(m) = P(\text{Binomial}(m, 1-p) > m/2) = \sum_{k > m/2} \binom{m}{k} (1-p)^k p^{m-k}$$

By Hoeffding's inequality: $P_{\text{error}}(m) \leq \exp(-2m(p - 1/2)^2)$. The error decays exponentially in m.

Example — Self-consistency sample size

Target error ≤ 0.01, per-path accuracy $p = 0.65$ ($p - 1/2 = 0.15$).

$$m \geq \frac{-\ln(0.01)}{2(0.15)^2} = \frac{4.605}{0.045} = 102.3$$

$m = 103$ samples suffice. For $p = 0.75$: $m \geq \frac{4.605}{2(0.25)^2} = 36.8$, so $m = 37$.
Practical note: the independence assumption is violated (same model, same prompt), but empirically self-consistency with $m = 20$–40 samples provides robust gains on MATH and GSM8K benchmarks.

59.6 AlphaProof and Lean Formal Verification

AlphaProof (DeepMind, 2024) combines LLM-generated proof sketches with a formal verifier (Lean 4).

Architecture.

1. Translate mathematical problem to Lean 4 (or Isabelle).
2. LLM generates candidate proof tactic sequences.
3. Lean type-checker verifies each step. Verification is decidable (Lean's type theory is consistent by construction up to large cardinals).
4. Successful proof steps used as training data (MCTS rollouts with verified reward).

Why verification matters. Without formal verification, LLM-generated mathematical reasoning has hallucination rates $\approx 15\%$ on competition problems (incorrect derivation steps presented confidently). With verification, each step either compiles or fails: no partial credit, no hallucination. AlphaProof achieved silver medal performance on IMO 2024 by combining this with a process reward model trained to predict step-level correctness.

PART 61 — SUPPLEMENT

Superposition: Tight Geometry, Polysemanticity and SAE Training

We derive the optimal superposition geometry (the regular polytope ansatz), analyse polysemanticity in MLP neurons, and discuss TopK SAEs as an improvement over L_1-penalised autoencoders.

61.4 Optimal Superposition Geometry

Regular Polytope Ansatz

For n features in d dimensions with equal importance and sparsity s, the optimal superposition geometry places feature vectors $\mathbf{w}_i$ at the vertices of a regular polytope in $\mathbb{R}^d$.

Derivation. The loss is $\mathcal{L} = \sum_i (1 - \|\mathbf{w}_i\|^2)^2 + \lambda s^2 \sum_{i \neq j} (\mathbf{w}_i^\top \mathbf{w}_j)^2$. At the minimum:

1. $\|\mathbf{w}_i\|^2 = 1$ for all i (unit norm).
2. $|\mathbf{w}_i^\top \mathbf{w}_j| = c$ for all $i \neq j$ (equal pairwise interference).

The second condition means $\{\mathbf{w}_i\}$ forms an *equiangular tight frame* (ETF). ETFs exist with up to $n = d(d+1)/2$ vectors in $\mathbb{R}^d$ (e.g. $d = 2$: up to 3 vectors forming an equilateral triangle; $d = 3$: up to 6 vectors at the vertices of an octahedron).

Gram matrix. For an ETF with n vectors in $\mathbb{R}^d$:

$$\mathbf{G}_{ij} = \mathbf{w}_i^\top \mathbf{w}_j = \delta_{ij} \pm \sqrt{\frac{n-d}{d(n-1)}}$$

The off-diagonal entries (interference) are $c = \sqrt{(n-d)/(d(n-1))}$, decreasing as $n \to d$ (fewer features) and $d \to n$ (more dimensions).

EXAMPLE — ETF geometry: 5 features in 2D

$n = 5, d = 2$: $c = \sqrt{(5-2)/(2(5-1))} = \sqrt{3/8} \approx 0.612$. Vectors point to the vertices of a regular pentagon. Pairwise cosines: $\cos(72°) \approx 0.309$, $\cos(144°) \approx -0.809$ — not equiangular. True ETFs in 2D can hold at most 3 equiangular vectors (equilateral triangle). For 5 features in 2D: the optimal packing is a pentagon but is not strictly equiangular, achieving $c \approx 0.31$.

61.5 Polysemanticity and Neuron Geometry

A neuron j in an MLP layer activates on multiple unrelated features (polysemanticity). This is a consequence of superposition: if feature $\mathbf{w}_i$ is spread across neurons as $\sum_j a_{ij} e_j$ (where e_j is the j-th neuron's direction), then neuron j activates for all features with $a_{ij} \neq 0$.

Polysemanticity index. For neuron j, the polysemanticity index $P_j = 1 - \sum_i a_{ij}^4 / (\sum_i a_{ij}^2)^2$. $P_j = 0$: monosemantic (activates for exactly one feature). $P_j \approx 1$: fully polysemantic (equal contribution from all features). Empirically, GPT-2 MLP neurons have $P_j \approx 0.8$–0.95 across layers, confirming widespread polysemanticity.

61.6 TopK SAEs

Standard SAE with L_1 penalty has issues: (a) L_1 shrinks all non-zero features toward zero (shrinkage bias); (b) the optimal λ varies with feature scale, requiring careful tuning.

TopK SAE. Replace the ReLU + L_1 with a hard TopK activation:

$$\mathbf{a} = \text{TopK}(\mathbf{W}_e(\mathbf{x} - \mathbf{b}_{\text{pre}}) + \mathbf{b}_e)$$

where $\text{TopK}(z)_i = z_i \cdot \mathbf{1}[z_i \geq z_{(K)}]$ (K largest retained, rest zeroed).

Advantages.

- No shrinkage: retained features are not penalised toward zero.
- Sparsity level K is a direct hyperparameter (vs. λ in L_1 SAE, which is indirect).
- Gradient: $d\text{TopK}(z)_i/dz_i = \mathbf{1}[z_i \geq z_{(K)}]$ (straight-through estimator for inactive units).

Gemma-Scope (Google DeepMind, 2024) deploys TopK SAEs with $K \in \{20, 40, 80\}$ features active per token for all layers and residual streams of Gemma 2 (2B–27B), making it the largest public release of interpretability artefacts to date.

PART 62 — SUPPLEMENT

Reward Hacking, Overoptimisation and the KL-Regularisation Theory

We derive the KL-reward tradeoff curve, analyse Goodhart's Law quantitatively in the context of RLHF, and derive the SimPO and ORPO objectives as reference-free alignment alternatives.

62.4 KL-Reward Tradeoff and Overoptimisation

Gao et al. (2023) empirically study how reward model score and gold reward evolve as the KL divergence $\mathrm{KL}(\pi_\theta||\pi_{\text{ref}})$ increases during RLHF. Define:

$$r_{\text{proxy}}(\pi) = \mathbb{E}_{y\sim\pi}[r_\phi(x,y)], \qquad r_{\text{gold}}(\pi) = \mathbb{E}_{y\sim\pi}[r^*(x,y)]$$

Empirical finding. r_{proxy} increases monotonically with KL (the proxy reward can always be increased by going further from the reference). r_{gold} initially increases but then decreases past a critical KL^*: this is overoptimisation (reward hacking).

Theoretical model. Assume the reward model has error ϵ: $r_\phi(x,y) = r^*(x,y) + \epsilon(x,y)$. The policy learns to exploit ϵ: as KL grows, it increasingly selects outputs where ϵ is large and positive, even if r^* is low.

KL constraint $\mathrm{KL} \le \delta$ ensures:

$$r_{\text{proxy}}(\pi) - r_{\text{gold}}(\pi) \le \|\epsilon\|_\infty \cdot \sqrt{2\delta} + O(\delta)$$

(by Pinsker's inequality: $|P(A) - Q(A)| \le \sqrt{\mathrm{KL}(P||Q)/2}$).

The optimal β (KL penalty coefficient) balances reward maximisation vs. error exploitation:

$$\beta^* \approx \frac{\|\epsilon\|_\infty}{\sqrt{2r_{\text{range}}}}$$

where r_{range} is the range of true rewards. Smaller reward model error $\|\epsilon\|$ allows larger β^* (more aggressive RL).

62.5 SimPO and Reference-Free Alignment

SimPO (Meng et al. 2024) eliminates the reference model by normalising rewards by sequence length and using a margin:

$$\mathcal{L}_{\text{SimPO}} = -\mathbb{E}\left[\log\sigma\left(\frac{\beta}{|y^+|}\log\pi_\theta(y^+|x) - \frac{\beta}{|y^-|}\log\pi_\theta(y^-|x) - \gamma\right)\right]$$

The length normalisation $1/|y|$ prevents the model from preferring short responses (which have higher log-probability under causal LMs) or long responses (which accumulate log-probabilities).

Connection to reward models. The length-normalised log-probability $\frac{1}{|y|}\log\pi(y|x)$ approximates the per-token average log-likelihood, which serves as a proxy for response quality without a separate reward model. The margin γ (typically 1.4) ensures a gap between preferred and rejected responses, analogous to SVM margin.

62.6 ORPO: Odds-Ratio Preference Optimisation

ORPO (Hong et al. 2024) combines supervised fine-tuning and preference alignment in a single objective:

$$\mathcal{L}_{\text{ORPO}} = \mathcal{L}_{\text{SFT}} - \lambda \cdot \mathbb{E}\left[\log \sigma\left(\log \frac{\text{odds}_\theta(y^+|x)}{\text{odds}_\theta(y^-|x)}\right)\right]$$

where $\text{odds}_\theta(y|x) = \pi_\theta(y|x)/(1-\pi_\theta(y|x))$ and $\mathcal{L}_{\text{SFT}} = -\log \pi_\theta(y^+|x)$.

Advantage over DPO. DPO requires a separate SFT phase followed by a preference learning phase. ORPO fuses them: the SFT term teaches the model to produce the preferred response while the odds-ratio term pushes it away from the rejected response. This is especially efficient when labelled preference pairs are limited.

PART 63 — SUPPLEMENT

Barren Plateaus, Quantum Advantage and Neuromorphic Hardware

We prove the barren plateau theorem for deep VQCs, survey the current landscape of quantum ML advantage claims, and analyse Intel Loihi-2 and IBM's neuromorphic architectures for spiking neural networks.

63.4 Barren Plateaus in VQCs

Barren Plateau Theorem (McClean et al. 2018)

For a VQC with L layers of 2-qubit gates applied to n qubits, if the architecture is a 2-design (sufficiently expressive), then the gradient variance satisfies:

$$\mathrm{Var}\left[\frac{\partial f}{\partial \theta_k}\right] \leq \frac{1}{2^n}$$

The gradient variance decays exponentially in the number of qubits n.

Proof sketch. For a circuit $U(\theta) = U_1 V U_2$ where U_1, U_2 are random and V is the gate containing θ_k, by the unitary 2-design property of U_2, the Haar-averaged gradient is:

$$\mathbb{E}\left[\left(\frac{\partial f}{\partial \theta_k}\right)^2\right] = \mathbb{E}[\langle\psi|O'|\psi\rangle^2] \leq \frac{\mathrm{tr}(O'^2)}{2^n(2^n+1)}$$

For a traceless observable O of bounded norm, $\mathrm{tr}(O'^2) \leq 1$, giving the exponential bound.

Implications. A VQC with $n = 50$ qubits has gradient variance $\leq 2^{-50} \approx 10^{-15}$ — numerically indistinguishable from zero. Training requires exponentially many circuit evaluations to achieve non-trivial signal-to-noise. This is the fundamental barrier to scaling VQCs beyond $\approx$ 10–20 qubits without structural inductive biases (e.g. symmetry-preserving ansätze).

63.5 Neuromorphic Hardware: Intel Loihi-2

Intel Loihi-2 (Davies et al. 2022) implements LIF neurons and STDP learning directly in silicon.

Key specifications.

- 1 million LIF neurons per chip, 120 million synapses.
- Event-driven (spike-based): computations only performed when spikes arrive.
- Energy: $\approx$ 10 pJ per spike, vs. $\approx 10^6$ pJ for an equivalent GPU matrix multiply of the same effective computation. Energy efficiency: $\sim 10^5\times$ better for sparse temporal workloads.

Effective operations. A LIF neuron receiving spikes at rate r_{in} performs approximately $r_{\text{in}} \times \tau_m$ additions per second (accumulating synaptic current). For $r_{\text{in}} = 100$ Hz and $\tau_m = 20$ ms: ≈ 2 additions per spike interval. A 1M-neuron chip with 1000 inputs per neuron and 100 Hz firing rate: 10^{11} synaptic operations/second at 10 pJ/op = 1 kW. A GPU achieving the same throughput: $10^{11} \times 10^{-6}$ J/op $= 10^5$ W. The comparison is approximate but illustrates the potential for $\sim 10^5\times$ energy savings in neuromorphic hardware for the right workloads.

PART 64 — SUPPLEMENT

Hardware-Aware NAS, Once-for-All and Neural Architecture Search at Scale

We derive the hardware-aware NAS multi-objective problem, analyse the Once-for-All (OFA) supernet approach, and discuss ProxylessNAS for direct gradient-based hardware latency minimisation.

64.4 Hardware-Aware NAS

The basic NAS objective minimises only validation loss. Hardware-aware NAS adds a latency (or energy) constraint:

$$\min_{a \in \mathcal{A}} \mathcal{L}_{\text{val}}(w^*(a), a) \quad \text{s.t.} \quad \text{Lat}(a, \text{device}) \leq T$$

where $\text{Lat}(a, \text{device})$ is the inference latency of architecture a on a target device.

Latency modelling. For a sequence of operations $o_1, \ldots, o_K$, the latency is approximately:

$$\text{Lat}(a) = \sum_{k=1}^{K} \text{lat}(o_k)$$

Each operation's latency $\text{lat}(o_k)$ is measured empirically on the target device and stored in a lookup table. This is a linear function of the architecture choices, enabling gradient computation:

$$\frac{\partial \text{Lat}(a)}{\partial \alpha_o^{(k)}} = \text{lat}(o) \cdot p_o^{(k)}(1 - p_o^{(k)})$$

where $p_o^{(k)} = \exp(\alpha_o^{(k)}) / \sum_{o'} \exp(\alpha_{o'}^{(k)})$ is the softmax probability of choosing operation o at layer k.

64.5 Once-for-All (OFA)

OFA (Cai et al. 2020) trains a single supernet that contains all sub-networks. Given a supernet with depth D, width multipliers $\{1.0, 1.2, 1.4\}$, and kernel sizes $\{3, 5, 7\}$:

- Total number of sub-networks: $(D_{\max}/D_{\min})^L \times 3^L \times 3^L$ where L is the number of layers.
- For $L = 20$, $D \in \{2, 3, 4\}$, width $\in \{3\}$, kernel $\in \{3, 5, 7\}$: $3^{20} \times 3^{20} \approx 10^{19}$ architectures.

Progressive shrinking. OFA trains the supernet progressively from large to small:

1. Train the full supernet at maximum depth/width/kernel.
2. Fine-tune with elastic kernel (simultaneously train kernel sizes $\{3, 5, 7\}$, sharing the inner 3×3 weights with the outer rings).
3. Fine-tune with elastic depth (simultaneously train all depth levels, using early-exit).
4. Fine-tune with elastic width (simultaneously train all width multipliers).

After training, any sub-network can be extracted and directly evaluated (no fine-tuning needed), enabling deployment across thousands of device configurations.

Example — OFA deployment for a mobile inference budget

Target: MobileNetV3-like architecture, latency ≤ 10 ms on Samsung Galaxy S20. After OFA training, search over 10^4 sampled architectures using the latency lookup table:

- Evaluate accuracy of each on a proxy dataset (200 images from ImageNet).
- Select the Pareto-optimal architecture maximising accuracy subject to ≤ 10 ms.

Result: 77.3% top-1 accuracy at 9.1 ms, vs. MobileNetV3-Large: 75.2% at 11.1 ms. OFA finds a strictly better accuracy-latency tradeoff than the hand-designed baseline.

PART 65 — SUPPLEMENT

VQ-VAE-2, Consistency Models and Latent Diffusion as Generative Unification

We derive the hierarchical VQ-VAE-2 latent structure, prove the consistency model distillation theorem, and show how latent diffusion reduces the dimensionality of the score matching problem.

65.6 VQ-VAE-2: Hierarchical Discrete Latents

VQ-VAE-2 (Razavi et al. 2019) introduces a hierarchy of two codebooks:

- **Bottom** codebook $\mathbf{E}^b \in \mathbb{R}^{K_b \times d_z}$: fine-grained local features (e.g. 8×8 spatial resolution for 256×256 images).
- **Top** codebook $\mathbf{E}^t \in \mathbb{R}^{K_t \times d_z}$: coarse global structure (4×4 resolution).

Architecture. Encoder: $\mathbf{z}_e^b = E_b(x)$, $\mathbf{z}_e^t = E_t(\mathbf{z}_e^b)$. Quantise: $\mathbf{z}_q^t = Q(\mathbf{z}_e^t)$, $\mathbf{z}_q^b = Q(\mathbf{z}_e^b | \mathbf{z}_q^t)$ (bottom quantisation conditioned on top). Decoder: $\hat{x} = D(\mathbf{z}_q^b, \mathbf{z}_q^t)$.

Generative model. After training, a PixelCNN-style autoregressive model is trained to generate sequences of discrete codes. The top codes are generated first (capturing global structure), then bottom codes conditioned on top (filling in detail). This two-stage generation allows VQ-VAE-2 to produce high-fidelity 256×256 images competitive with GANs at the time.

65.7 Consistency Models

Consistency models (Song et al. 2023) distil a diffusion model into a one- or few-step sampler.

Consistency Function

A consistency model is a function $f_\theta : (\mathbf{x}_t, t) \mapsto \mathbf{x}_0$ satisfying:

$$f_\theta(\mathbf{x}_t, t) = f_\theta(\mathbf{x}_{t'}, t') \quad \forall t, t' \text{ on the same PF ODE trajectory}$$

("consistency property"). The boundary condition: $f_\theta(\mathbf{x}_0, 0) = \mathbf{x}_0$.

Consistency distillation objective. Given a pretrained diffusion model score s_ϕ and adjacent times $t_n < t_{n+1}$, estimate $\mathbf{x}_{t_n}$ by one ODE step:

$$\hat{\mathbf{x}}_{t_n}^{(s_\phi)} = \mathbf{x}_{t_{n+1}} + (t_n - t_{n+1}) s_\phi(\mathbf{x}_{t_{n+1}}, t_{n+1})$$

The consistency distillation loss enforces:

$$\mathcal{L}_{\text{CD}} = \mathbb{E}\Big[d\Big(f_\theta(\mathbf{x}_{t_{n+1}}, t_{n+1}),\ f_{\theta^-}(\hat{\mathbf{x}}_{t_n}^{(s_\phi)}, t_n) \Big) \Big]$$

where θ^- is an EMA (exponential moving average) of θ (stop-gradient), and $d(\cdot,\cdot)$ is a distance metric (LPIPS in practice).

One-step generation. Once trained, $f_\theta(\mathbf{x}_T, T)$ maps pure noise $\mathbf{x}_T \sim \mathcal{N}(0, T^2 \mathbf{I})$ directly to $\mathbf{x}_0$ in a single forward pass. FID on CIFAR-10 (single step): 3.55 vs. EDM diffusion (35 steps): 2.04. With 2 steps: 2.93 — near-diffusion quality at 17× speedup.

65.8 Latent Diffusion Models (LDMs)

LDMs (Rombach et al. 2022) — the architecture underlying Stable Diffusion — apply diffusion in a compressed latent space rather than pixel space.

Two-stage framework.

1. **Perceptual compression.** Train a VQ-VAE or KL-VAE encoder-decoder $(\mathcal{E}, \mathcal{D})$ to compress images $x \in \mathbb{R}^{H\times W\times 3}$ to latents $z = \mathcal{E}(x) \in \mathbb{R}^{h\times w\times c}$ with compression factor $f = H/h = W/w$. Stable Diffusion uses $f = 8$: 512×512 images $\to 64 \times 64 \times 4$ latents.

2. **Latent diffusion.** Apply the DDPM/score-matching framework in the latent space:

$$\mathcal{L}_{\text{LDM}} = \mathbb{E}_{z\sim\mathcal{E}(x),\epsilon\sim\mathcal{N}(0,\mathbf{I}),t}\left[\|\epsilon - \epsilon_\theta(z_t, t, \tau_\psi(y))\|^2\right]$$

   where $\tau_\psi(y)$ is the conditioning signal (text embedding from CLIP).

Dimensionality reduction benefit. The latent space is $64 \times 64 \times 4 = 16384$ dimensions vs. $512 \times 512 \times 3 = 786432$ pixel dimensions. Score network complexity $\propto d^2$ (attention): reduction by $(786432/16384)^2 = 2300\times$ in naive attention cost. In practice, the U-Net with cross-attention is $\sim 50\times$ cheaper to train and sample in the latent space than in pixel space.

PART 66

Test-Time Training, LoRA and Parameter-Efficient Fine-Tuning

We derive the mathematics of Low-Rank Adaptation (LoRA), analyse the intrinsic dimension of fine-tuning tasks, and develop the theory of test-time training as an online adaptation protocol.

66.1 Intrinsic Dimension of Fine-Tuning

Li et al. (2018) show that fine-tuning large language models requires navigating only a low-dimensional subspace of the parameter space.

Intrinsic Dimension

A fine-tuning task has *intrinsic dimension* d_{90} if optimising over d_{90} random linear combinations of the full parameter vector $\square \in \mathbb{R}^D$ achieves 90% of the performance of full fine-tuning. Formally:

$$\square = \square_0 + \mathbf{P}\square, \quad \square \in \mathbb{R}^{d_{90}} \ll D, \quad \mathbf{P} \in \mathbb{R}^{D \times d_{90}} \text{ fixed random}$$

Empirical findings: BERT-base ($D = 110M$ parameters) has $d_{90} \approx 200$ for MRPC (paraphrase detection) and $d_{90} \approx 800$ for CoLA (grammaticality judgement). GPT-2 large ($D = 774M$) has $d_{90} \approx 1000$ for most GLUE tasks. This suggests that the effective degrees of freedom for fine-tuning are far fewer than the nominal parameter count.

66.2 LoRA: Low-Rank Adaptation

LoRA (Hu et al. 2022) parameterises weight updates as low-rank matrices:

$$W = W_0 + \Delta W = W_0 + BA, \qquad B \in \mathbb{R}^{d \times r},\ A \in \mathbb{R}^{r \times k},\ r \ll \min(d, k)$$

W_0 is frozen; A is initialised as $\mathcal{N}(0, \sigma^2/r)$; B initialised to zero (so $\Delta W = 0$ at start).

Gradient analysis. During fine-tuning, the gradient w.r.t. W_0 in full fine-tuning is $G = \dfrac{\partial \mathcal{L}}{\partial W_0} \in \mathbb{R}^{d \times k}$. The LoRA approximation assumes $\operatorname{rank}(G) \le r$. The optimal low-rank approximation to G (by Eckart-Young theorem):

$$G_r = U_r \Sigma_r V_r^\top = \arg\min_{\operatorname{rank}(M) \le r} \|G - M\|_F$$

with U_r, V_r from the SVD of G. LoRA learns B, A to approximate G_r.

Rank selection. The fraction of variance captured by rank-r approximation:

$$\rho(r) = \frac{\sum_{i=1}^{r} \sigma_i^2}{\sum_{i=1}^{d} \sigma_i^2}$$

For stable rank $\operatorname{srank}(G) = \|G\|_F^2 / \|G\|_2^2$: $\rho(r) \ge 1 - \operatorname{srank}(G)/r$. Tasks with low stable rank (few dominant directions of gradient) benefit most from LoRA.

Example — LoRA parameter count reduction, LLaMA-7B, $r = 8$

Attention weight dimensions: $W_Q, W_K, W_V, W_O \in \mathbb{R}^{4096\times4096}$. Parameters per matrix: $4096^2 = 16.7M$.
LoRA per matrix: $B \in \mathbb{R}^{4096\times8}$ (32K) + $A \in \mathbb{R}^{8\times4096}$ (32K) = 64K.
Reduction per matrix: $16.7M/64K = 261\times$.
Total LoRA parameters (32 layers, 4 matrices): $32 \times 4 \times 64K = 8.2M$ vs. full fine-tuning $7B$. LoRA fine-tunes 0.12% of the parameters while achieving competitive performance on instruction following.

66.3 QLoRA: 4-bit Quantisation with LoRA

QLoRA (Dettmers et al. 2023) combines LoRA with 4-bit NormalFloat (NF4) quantisation.

NF4 quantisation. For a weight tensor $\mathbf{w}$ assumed to be normally distributed $\mathcal{N}(0,1)$ after rescaling, NF4 defines $2^4 = 16$ quantisation levels as the quantiles of $\mathcal{N}(0,1)$:

$$q_i = \Phi^{-1}\left(\frac{i}{2^4-1}\right), \quad i = 0, \ldots, 15$$

where Φ^{-1} is the normal quantile function. This is information-theoretically optimal for normally distributed data: each of the 16 bins contains equal probability mass, minimising expected quantisation error.

Double quantisation. Quantisation constants (scale factors) are themselves quantised (8-bit), reducing memory by an additional ~ 0.37 bits/parameter.

Memory calculation. LLaMA-65B in full precision: $65B \times 4$ bytes $= 260$ GB. QLoRA: $65B \times 0.5$ bytes (4-bit) $= 32.5$ GB + LoRA adapters (~ 300 MB) ≈ 33 GB. Fits on a single 40 GB A100 — enabling fine-tuning of 65B models on single-GPU setups previously impossible.

66.4 Test-Time Training (TTT)

TTT (Sun et al. 2020; revisited by Akyürek et al. 2024) performs gradient updates on the model using the test input itself, creating a per-instance adapted model.

Protocol. For a test instance x_*:

1. Define a self-supervised auxiliary task $\mathcal{T}(x_*)$ (e.g. masked reconstruction, rotation prediction).
2. Compute K gradient steps: $\theta^{(k+1)} = \theta^{(k)} - \eta\nabla_\theta \mathcal{L}_\mathcal{T}(x_*)$.
3. Predict with the adapted model: $\hat{y} = f_{\theta^{(K)}}(x_*)$.

Convergence analysis. Under smoothness assumption $\|\nabla^2\mathcal{L}\| \le L$, after K steps:

$$\mathcal{L}_\mathcal{T}(\theta^{(K)}) \le \mathcal{L}_\mathcal{T}(\theta^*) + \frac{L\|\theta^{(0)} - \theta^*\|^2}{2K\eta}$$

The adaptation error is controlled by the distance to the optimal per-instance parameters θ^* and the number of steps K. For in-distribution inputs, $\|\theta^{(0)} - \theta^*\|$ is small (meta-learning provides a good initialisation); for out-of-distribution inputs, more steps are needed.

TTT layers. Akyürek et al. (2024) propose replacing a subset of attention layers with "TTT layers": at each token position, a mini-gradient-descent step is performed on a small "fast

weights" network, enabling the model to adapt its internal representations dynamically within a single forward pass. This is equivalent to running SGD inside the transformer, blurring the boundary between inference and training.

PART 67

Multimodal Architectures: CLIP, Flamingo and Vision Transformers

We derive the CLIP contrastive loss, analyse the patch embedding construction of Vision Transformers (ViT), and develop the cross-attention conditioning mechanism of Flamingo for visual language models.

67.1 CLIP: Contrastive Language-Image Pre-training

CLIP (Radford et al. 2021) trains image and text encoders jointly to align their representations.

CLIP Contrastive Loss

For a batch of N image-text pairs $\{(I_i, T_i)\}_{i=1}^{N}$, the CLIP loss is:

$$\mathcal{L}_{\text{CLIP}} = \frac{1}{2N}\sum_{i=1}^{N}\left[\mathcal{L}^{(i)}_{\text{img}\to\text{txt}} + \mathcal{L}^{(i)}_{\text{txt}\to\text{img}}\right]$$

$$\mathcal{L}^{(i)}_{\text{img}\to\text{txt}} = -\log\frac{\exp(\mathbf{f}_i^\top \mathbf{g}_i/\tau)}{\sum_{j=1}^{N}\exp(\mathbf{f}_i^\top \mathbf{g}_j/\tau)}$$

where $\mathbf{f}_i = F_I(I_i)/\|F_I(I_i)\|$, $\mathbf{g}_i = F_T(T_i)/\|F_T(T_i)\|$, and $\tau > 0$ is a learned temperature.

Interpretation. This is InfoNCE (mutual information maximisation) for image-text pairs. With $N = 32768$ (CLIP's batch size), the model solves a 32768-way classification problem: given an image, identify its matching caption from 32767 negatives.

Optimal temperature. The temperature τ controls the sharpness of the distribution. At $\tau \to 0$: the loss reduces to finding the single best match (argmax). At $\tau \to \infty$: the distribution is uniform (no learning signal). The optimal τ is related to the concentration of the representation space: $\tau^* \approx 1/\sqrt{d}$ for d-dimensional unit sphere representations.

Zero-shot transfer. Given a downstream classification task with classes $c_1, \ldots, c_K$, CLIP creates text prompts "a photo of a c_k" and classifies by:

$$\hat{k} = \arg\max_k \mathbf{f}(I)^\top \mathbf{g}(\text{"a photo of a } c_k\text{"})$$

No fine-tuning needed. ImageNet top-1 accuracy: 76.2% (CLIP ViT-L/14) vs. ResNet-50 supervised: 76.1%. CLIP achieves comparable accuracy to supervised ImageNet training *without ever seeing an ImageNet label*.

67.2 Vision Transformer (ViT)

ViT (Dosovitskiy et al. 2021) applies the standard Transformer directly to image patches.

Patch Embedding

Given image $I \in \mathbb{R}^{H\times W\times C}$, divide into $N_p = (H/P)(W/P)$ patches of size $P \times P \times C$. Flatten each patch to a vector $\mathbf{p}_i \in \mathbb{R}^{P^2C}$ and project:

$$\mathbf{z}_i = \mathbf{W}_E\mathbf{p}_i + \mathbf{b}_E, \quad \mathbf{W}_E \in \mathbb{R}^{d\times P^2C}$$

Prepend a learnable classification token $\mathbf{z}_0 = \mathbf{x}_{\text{cls}}$. Add positional embeddings.

Attention in ViT. Each patch attends to all other patches: $O(N_p^2 d)$ cost. For $P = 16$, $H = W = 224$: $N_p = 196$ patches. For $P = 4$, $H = W = 224$: $N_p = 3136$ patches — quadratic attention becomes expensive at small patch sizes.

ViT scaling. ViT-Huge: $L = 32$ layers, $d = 1280$, $H = 16$ heads, $632M$ parameters. Trained on JFT-3B (3 billion image-label pairs), achieves 90.5% top-1 on ImageNet with heavy augmentation. Key finding: ViT is more data-hungry than ResNet at small datasets but scales better at large scale ($> 300M$ examples).

67.3 Flamingo: Vision-Language Models via Cross-Attention

Flamingo (Alayrac et al. 2022) combines a pretrained vision encoder (NFNet) and a pretrained language model (Chinchilla-70B) via cross-attention layers inserted at every 6th transformer layer.

Flamingo Cross-Attention Layer

For language token representations $\{\mathbf{x}_i\}$ and visual features $\{\mathbf{v}_j\}$ (from the vision encoder + Perceiver resampler):

$$\mathbf{x}_i' = \mathbf{x}_i + \tanh(\alpha) \cdot \text{CrossAttn}(\mathbf{x}_i, \{\mathbf{v}_j\})$$

$$\text{CrossAttn}(\mathbf{x}_i, \{\mathbf{v}_j\}) = \sum_j \text{softmax}\left(\frac{(\mathbf{W}_Q \mathbf{x}_i)^\top (\mathbf{W}_K \mathbf{v}_j)}{\sqrt{d_k}}\right) \mathbf{W}_V \mathbf{v}_j$$

α is initialised to 0 (visual conditioning inactive), learned to gradually incorporate visual information.

Perceiver resampler. The vision encoder produces $O(N_p)$ features (variable by image resolution). A Perceiver resampler with $m = 64$ fixed learned queries attends to the visual features, producing exactly m conditioning vectors regardless of image size:

$$\tilde{\mathbf{v}}_j = \sum_k \text{softmax}\left(\frac{(\mathbf{W}_Q \mathbf{q}_j)^\top (\mathbf{W}_K \mathbf{v}_k)}{\sqrt{d_k}}\right) \mathbf{W}_V \mathbf{v}_k$$

where $\mathbf{q}_j$ are the learned query vectors. This bounds the cross-attention cost at $O(m)$ per language token, independent of image resolution.

Part 68

Diffusion Models: Score Matching, DDPM and Flow Matching

We provide a complete derivation of the denoising score matching objective, prove the DDPM evidence lower bound, and derive the conditional flow matching (CFM) vector field from optimal transport.

68.1 Score Matching

The score function is $\nabla_x \log p(x)$. Score matching (Hyvärinen, 2005) trains a model $s_\theta(x) \approx \nabla_x \log p(x)$ without computing the partition function.

Implicit score matching objective. Minimising $\mathcal{L}_{\text{SM}} = \mathbb{E}_p\big[\frac{1}{2}\|s_\theta(x)\|^2 + \operatorname{tr}(\nabla_x s_\theta(x))\big]$ is equivalent (up to constants independent of θ) to minimising $\mathbb{E}_p[\|s_\theta(x) - \nabla_x \log p(x)\|^2/2]$.

Proof. Expand the explicit objective $J_{\text{ESM}} = \mathbb{E}_p[\|s_\theta - \nabla \log p\|^2/2]$:

$$J_{\text{ESM}} = \mathbb{E}_p\left[\frac{1}{2}\|s_\theta\|^2\right] - \mathbb{E}_p[s_\theta \cdot \nabla \log p] + \text{const}$$

The cross term: $\mathbb{E}_p[s_\theta \cdot \nabla \log p] = \int p(x) s_\theta(x) \cdot \frac{\nabla p(x)}{p(x)} dx = \int s_\theta(x) \cdot \nabla p(x) dx$. Integration by parts: $= -\int p(x) \operatorname{tr}(\nabla_x s_\theta(x))\, dx = -\mathbb{E}_p[\operatorname{tr}(\nabla_x s_\theta)]$. Substituting: $J_{\text{ESM}} = \mathbb{E}_p[\|s_\theta\|^2/2 + \operatorname{tr}(\nabla_x s_\theta)] + \text{const} = J_{\text{ISM}} + \text{const}$. $\square$

68.2 DDPM Derivation

DDPM (Ho et al. 2020) defines a forward noising process and trains a reverse denoising process.

Forward process. $q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t \mathbf{I})$. By induction:

$$q(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)\mathbf{I}), \qquad \bar{\alpha}_t = \prod_{s=1}^{t}(1-\beta_s)$$

So $x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon$, $\epsilon \sim \mathcal{N}(0, \mathbf{I})$.

Reverse process ELBO. The VLB is:

$$-\log p_\theta(x_0) \le \mathbb{E}_q\left[\sum_{t>1} D_{\text{KL}}(q(x_{t-1}|x_t, x_0)\|p_\theta(x_{t-1}|x_t)) - \log p_\theta(x_0|x_1)\right]$$

The posterior $q(x_{t-1}|x_t, x_0)$ is Gaussian with mean:

$$\tilde{\mu}_t(x_t, x_0) = \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}x_0 + \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}x_t$$

Simplified objective. Ho et al. show that parameterising $p_\theta(x_{t-1}|x_t)$ as predicting the noise $\epsilon_\theta(x_t, t)$ and using the simplified loss:

$$\mathcal{L}_{\text{simple}} = \mathbb{E}_{x_0,\epsilon,t}\left[\|\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, t)\|^2\right]$$

outperforms the full ELBO in sample quality, despite not being a proper bound.

68.3 Conditional Flow Matching

CFM (Lipman et al. 2023) learns a vector field $v_\theta(x,t)$ that transports source distribution $p_0 = \mathcal{N}(0,\mathbf{I})$ to data distribution p_1.

Flow Matching Objective

Given a probability path p_t interpolating $p_0 \to p_1$ and the associated marginal vector field $u_t(x)$:

$$\mathcal{L}_{\text{CFM}} = \mathbb{E}_{t,q(x_1),p_t(x|x_1)}\left[\|v_\theta(x,t) - u_t(x|x_1)\|^2\right]$$

where the conditional vector field $u_t(x|x_1)$ generates the path $p_t(x|x_1)$ for a single data point x_1.

Optimal transport conditional path. Choose $p_t(x|x_1) = \mathcal{N}(tx_1, (1-t)^2\mathbf{I})$ (linear interpolation with shrinking noise). The conditional vector field:

$$u_t(x|x_1) = \frac{x_1 - (1-t)\epsilon}{1 - t + t\epsilon/\|x_1 - x_0\|} \approx \frac{x_1 - x}{1-t}$$

simplifies in the Gaussian case to $u_t(x|x_1) = (x_1 - x)/(1-t)$. This is a simple closed-form target, requiring no diffusion ELBO or score network.

ODE integration. At inference, numerically integrate:

$$\frac{dx}{dt} = v_\theta(x,t), \qquad x(0) \sim \mathcal{N}(0,\mathbf{I}), \quad \text{output } x(1)$$

With N Euler steps: $x_{k+1} = x_k + v_\theta(x_k, k/N)/N$. The Lipschitz constant of v_θ bounds the discretisation error: $\|x^{(N)}(1) - x(1)\| \le L/(2N)$ (first-order Euler).

Part 69

Open Problems and Mathematical Frontiers in Deep Learning

We survey the major unresolved mathematical questions in deep learning: neural network expressivity limits, the implicit bias of gradient descent, benign overfitting theory, and the role of symmetry in foundation models.

69.1 The Implicit Bias of Gradient Descent

When a neural network perfectly interpolates training data (zero training loss), the gradient descent trajectory determines which interpolant is chosen. This implicit bias is a form of regularisation not explicitly specified in the loss.

Linear networks: minimum-norm interpolation. For a linear model $f(x) = Wx$ trained to interpolate n examples $\{(x_i, y_i)\}$ with $n < d$ (underdetermined), gradient descent from $W_0 = 0$ converges to the minimum-$\|W\|_F$ solution:

$$W^* = YX^\top (XX^\top)^{-1}$$

This is the Moore-Penrose pseudoinverse solution. Proof: $W(t) = \sum_i c_i(t) y_i x_i^\top$ (always in the row span of X), so the minimum norm solution is achieved.

Deep linear networks: nuclear norm bias. For a depth-2 linear network $W = W_2 W_1$, gradient descent from small random initialisation converges to the minimum *nuclear norm* solution:

$$W^* = \arg\min_{WX=Y} \|W\|_* = \sum_i \sigma_i(W)$$

This is a stronger regularisation than Frobenius norm: it promotes low-rank solutions. Empirical evidence suggests that deep non-linear networks also have a low-rank implicit bias, explaining the effectiveness of LoRA.

69.2 Benign Overfitting

Benign Overfitting (Bartlett et al. 2020)

A model is a benign overfitter if it interpolates noisy training data (zero training error on corrupted labels) yet achieves near-optimal test error.

Condition for benign overfitting. For linear models with covariance $\Sigma = \mathrm{diag}(\sigma_1^2, \ldots, \sigma_d^2)$ and n training examples, benign overfitting occurs when:

$$\frac{\sum_{j>k} \sigma_j^2}{\max_j \sigma_j^2} \gg n \qquad \text{(the tail of the spectrum dominates)}$$

i.e. there are very many small-variance directions relative to the few large-variance signal directions. The minimum-norm interpolant uses the tail directions to fit the noise, while the signal directions are largely unaffected.

Transformer connection. Overparameterised transformers operate in a regime where the effective dimension of the loss landscape's Hessian has a heavy-tailed spectrum (empirical evidence from spectral analysis of $\nabla^2 \mathcal{L}$). This may explain why transformers can memorise entire training corpora while retaining strong generalisation: the memorisation occurs in the spectral tail, leaving the top eigenspaces (which drive generalisation) intact.

69.3 Equivariance as a Design Principle

The most powerful inductive biases in modern architectures are *equivariance constraints*: the architecture's output transforms predictably under input symmetries.

G-equivariance

A function $f : \mathcal{X} \to \mathcal{Y}$ is G-equivariant if for all $g \in G$ and $x \in \mathcal{X}$:

$$f(\rho_{\mathcal{X}}(g)x) = \rho_{\mathcal{Y}}(g)f(x)$$

where $\rho_{\mathcal{X}}, \rho_{\mathcal{Y}}$ are representations of the group G on the input and output spaces.

Examples.

- CNNs: translation equivariance ($G = \mathbb{Z}^2$, ρ = spatial shift).
- E(3)-equivariant networks (e.g. SE(3)-Transformer, SEGNN): equivariant to 3D rotations and translations. Critical for molecular property prediction.
- Attention: permutation equivariance ($G = S_n$, the symmetric group). Adding positional encoding breaks this symmetry — the attention mechanism is fundamentally permutation-equivariant without positional information.

Schur's lemma and architectural constraints. For an equivariant linear map $W : V \to V$ where V is an irreducible representation of G, Schur's lemma states $W = \lambda \mathbf{I}$ for some $\lambda \in \mathbb{R}$. This drastically constrains the architecture: equivariant layers have far fewer parameters than generic linear layers. For the rotation group $SO(3)$ acting on vectors ($d = 3$): only 1 scalar degree of freedom (vs. 9 for a general 3×3 matrix).

69.4 The Depth Separation Problem

A central open problem: do deep networks provably require fewer neurons than shallow networks to approximate a given class of functions?

Known results.

- Telgarsky (2016): there exist functions computable by a depth-t ReLU network with $\Theta(n)$ neurons but requiring $2^{\Omega(n^{1/t})}$ neurons to approximate with depth $\leq t-1$. Depth separation is exponential.
- Eldan-Shamir (2016): there exist functions approximable in depth-3 with $O(d)$ neurons but requiring $2^{\Omega(d)}$ neurons in depth-2 (with d input dimensions).
- Transformers vs. RNNs: for periodic tasks (modular arithmetic), transformers can solve in $O(1)$ layers what RNNs require $O(n)$ steps.

Open question. Is there a function class where transformers provably outperform SSMs (Mamba-style architectures), or vice versa? Current evidence is empirical: transformers excel at tasks requiring global comparison (retrieval, reasoning); SSMs excel at streaming/online tasks. A formal complexity separation remains open.

69.5 Foundation Models and Emergent World Models

The central mystery of large language models is the degree to which they build *world models* — internal representations of causal structure in the data, not just surface-level correlations.

Linear representation hypothesis. Probing experiments (Nanda et al. 2023; Park et al. 2024) suggest that world-model features are encoded *linearly* in the residual stream: there exist directions $\mathbf{d} \in \mathbb{R}^d$ such that $\mathbf{d}^\top h \approx f(x)$ for world-model features f (e.g. position on a map, days of the week, truth value of a proposition).

Causal interventions. If $\mathbf{d}$ linearly encodes day-of-week, then adding $\Delta \cdot \mathbf{d}$ to the residual stream at a given layer should shift the model's output as if the day were shifted by Δ units. Experiments confirm this for many features, suggesting that transformers encode a structured, linearly manipulable world model.

Open problem: grounding. How do these linear representations arise from next-token prediction alone? A theoretical explanation would require understanding how the self-supervised objective on natural language creates pressure to develop structured internal representations, rather than relying on arbitrary implicit encodings. This is arguably the central open problem in mechanistic interpretability.

> **The open frontier.** Volume V has traced the mathematical arc from efficient attention ($O(n\sqrt{n})$ through to $O(n)$) through state space models, scaling laws, emergence, alignment, and interpretability. The field's open problems are not technical gaps but foundational questions: Why does gradient descent have an implicit low-rank bias? Under what conditions is benign overfitting provably safe? Do transformers develop genuine causal world models, and if so, why? The mathematical tools developed across these five volumes — linear algebra, probability, differential geometry, group theory, information theory, statistical learning theory — are exactly the tools needed to attack these questions. The frontier begins here.

Appendix: Mathematical Reference Symbols

Symbol	Definition
$\mathbf{A}, \mathbf{B}, \mathbf{C}$	State space model matrices (continuous-time)
$\bar{\mathbf{A}}, \bar{\mathbf{B}}$	Discretised SSM matrices (zero-order hold)
α, β	Chinchilla scaling exponents ($\alpha \approx 0.34$, $\beta \approx 0.28$)
$\bar{\alpha}_t$	Cumulative noise schedule product in DDPM
$\mathbf{B}, \mathbf{A}$	LoRA low-rank factor matrices ($\mathbb{R}^{d\times r}$, $\mathbb{R}^{r\times k}$)
β	RLHF KL penalty coefficient; also slope in ALiBi
$c_n(t)$	HiPPO polynomial projection coefficients
D	Training token count; also decoder function
d_{90}	Intrinsic dimension of a fine-tuning task
Δ	Discretisation step size in SSMs; also LoRA rank
E	Irreducible entropy in Chinchilla loss formula
$\mathcal{E}$	Encoder function; also approximation error
ϵ_θ	Noise prediction network in DDPM
$\mathbf{E}^b, \mathbf{E}^t$	Bottom and top VQ-VAE-2 codebooks
f	Firing rate (LIF neurons); fraction of tokens (MoE)
$\phi^+(x)$	FAVOR+ positive random features
γ	xPos decay factor per dimension; also ramp function
G	Number of GQA key-value groups; also gradient matrix
H	Number of attention heads
$\mathbf{h}(t)$	Hidden state of continuous-time SSM
K	TopK sparsity level; also number of codebook entries
κ	Condition number $\lambda_{\text{max}}/\lambda_{\text{min}}$
λ_i	Eigenvalues of SSM matrix $\mathbf{A}$; RoPE wavelength
$\mathbf{L}_{ts}$	Scalar decay product $\prod_{r=s+1}^{t} a_r$ in SSD
$\mathcal{L}_{\text{DPO}}$	Direct preference optimisation loss
m_h	ALiBi head-specific slope
$\mathbf{M}$	Semiseparable masking matrix in SSD
N	Number of model parameters; SSM state dimension
n	Sequence length; also number of tokens
P_j	Polysemanticity index of neuron j
$\pi_\theta, \pi_{\text{ref}}$	RLHF policy and reference policy
q_i, k_j, v_j	Query, key, value vectors at position i, j
r	LoRA rank; also LSH hash function count
r_ϕ	Reward model
$\rho(r)$	Fraction of variance captured by rank-r approximation
$s_\theta(x)$	Score network approximating $\nabla_x \log p(x)$
τ	CLIP temperature; also membrane time constant
θ_i	RoPE rotation frequency for dimension pair i
$u_t(x\|x_1)$	Conditional vector field in flow matching
$v_\theta(x,t)$	Learned vector field in CFM
$\mathbf{W}_E$	ViT patch embedding projection; LM token embedding
w, W	Sliding window size; attention window
$\mathbf{z}_e, \mathbf{z}_q$	VQ-VAE encoder output and quantised latent

PART 53 — EXTENDED SUPPLEMENT

Multi-Head Latent Attention, Sparse Attention and KV Cache Compression

We derive Multi-Head Latent Attention (MLA) as used in DeepSeek, analyse the mathematics of key-value cache compression via low-rank projection, and develop the theory of block-sparse attention patterns with guaranteed approximation quality.

53.7 Multi-Head Latent Attention (MLA)

MLA (DeepSeek-V2, 2024) compresses the KV cache by projecting keys and values through a shared low-dimensional latent:

MLA Projection Scheme

Given input $\mathbf{x} \in \mathbb{R}^d$, compute a shared KV latent:

$$\mathbf{c}_{KV} = \mathbf{W}_{DKV}\mathbf{x} \in \mathbb{R}^{d_c}, \qquad d_c \ll d \cdot H$$

Expand to per-head keys and values:

$$\mathbf{K} = \mathbf{W}_{UK}\mathbf{c}_{KV}, \quad \mathbf{V} = \mathbf{W}_{UV}\mathbf{c}_{KV}$$

with $\mathbf{W}_{UK} \in \mathbb{R}^{Hd_k \times d_c}$, $\mathbf{W}_{UV} \in \mathbb{R}^{Hd_v \times d_c}$.

KV cache memory. Standard MHA caches $\mathbf{K}, \mathbf{V} \in \mathbb{R}^{n \times Hd_k}$, requiring $2nHd_k$ values per layer. MLA instead caches only $\mathbf{c}_{KV} \in \mathbb{R}^{d_c}$ per position, requiring nd_c values. With $d_c = 512$, $H = 128$, $d_k = 128$: compression ratio $= 2 \times 128 \times 128/512 = 64\times$.

Decoupled RoPE. A subtlety: RoPE must be applied to queries and keys after expansion, but the compressed latent $\mathbf{c}_{KV}$ must be cached *without* RoPE (since position only becomes known at query time). MLA handles this by maintaining two components in the key: an RoPE-rotated part and a non-RoPE part:

$$k_i = [k_i^{\text{noRoPE}};\ \mathbf{R}_m k_i^{\text{RoPE}}]$$

where k_i^{noRoPE} is derived from $\mathbf{c}_{KV}$ and k_i^{RoPE} is derived from a separate projection of $\mathbf{x}$. Only $\mathbf{c}_{KV}$ and k_i^{RoPE} are cached.

Parameter efficiency. MLA's total KV parameter count is $\mathbf{W}_{DKV}$ $(\mathrm{d}_c \times d) plus \mathrm{W}_{UK}, \mathbf{W}_{UV}$ $(Hd_kd_c + Hd_vd_c)$. The up-projection matrices can be absorbed into the query and output projections during inference, eliminating the latent bottleneck cost entirely.

EXAMPLE — MLA memory vs. MHA for DeepSeek-V2, 128 heads, $d = 7168$, $d_k = 128$

MHA KV cache per token per layer: $2 \times 128 \times 128 = 32{,}768$ values = 65.5 KB (fp16).
MLA with $d_c = 512$: 512 values = 1.0 KB (fp16). Compression: $64\times$.
For a 60-layer model with 128K context: MHA: $60 \times 128 \times 10^3 \times 65.5\,\text{KB} = 500\,\text{GB}$ (impossible on a single H100). MLA: $60 \times 128 \times 10^3 \times 1.0\,\text{KB} = 7.5\,\text{GB}$ (fits easily).

53.8 Sparse Attention: Block-Sparse Patterns

A block-sparse attention pattern partitions the sequence into blocks of size B and attends only between selected block pairs. Formally, define a block index set $\mathcal{S} \subseteq \{(b_q, b_k) : 0 \leq$

$b_q, b_k < n/B\}$:

$$A_{ij} \neq 0 \iff (\lfloor i/B \rfloor, \lfloor j/B \rfloor) \in \mathcal{S}$$

If $|\mathcal{S}| = s$ block pairs, the total non-zero attention entries are sB^2 vs. n^2 for full attention.

BigBird sparsity. BigBird's pattern combines:

- n/B diagonal block pairs (sliding window): nB entries.
- G global token pairs (every token attends to G globals and vice versa): $2nG$ entries.
- r random pairs per token: nrB entries.

Total non-zeros: $O(n(B + G + rB))$, linear in n for fixed B, G, r.

Approximation quality. For a sequence where the true attention matrix $\mathbf{A}$ is (ϵ, k)-concentrated (each row has k entries accounting for $(1 - \epsilon)$ fraction of the row's weight), block-sparse attention with block size $B = k$ and appropriate pattern $\mathcal{S}$ approximates $\mathbf{A}$ up to ϵ error in each output token. The random attention edges provide a coverage guarantee: with $r = O(\log n)$ random edges per token, every pair of tokens has an ℓ-hop path of length $O(\log n)$ with high probability, enabling information propagation across the sequence.

53.9 KV Cache Compression via Streaming

For autoregressive generation with long context, the full KV cache cannot always fit in GPU memory. Two complementary strategies:

StreamingLLM. Xiao et al. (2023) observe that a small set of "sink" tokens at the beginning of the sequence receive disproportionate attention regardless of content. The attention score distribution: let $\text{sink}(j)$ be the attention allocated to position j by position $i > j$. Empirically, $\text{sink}(1), \text{sink}(2) \gg \text{sink}(j)$ for $j > 2$ for early positions, even when position 1 is semantically irrelevant.

Why sink tokens? RoPE encodes position 0 as a rotation by 0 radians, making $\mathbf{R}_0 = \mathbf{I}$. The key at position 0 is never rotated, making it a "default" key that all queries have non-zero inner product with. Without explicit structure, the softmax assigns residual probability mass to these positions. The "sink" phenomenon is an artefact of the rotation structure, not semantic content.

H2O (Heavy Hitter Oracle). Zhang et al. (2023) evict KV cache entries greedily: at each step, retain the k entries with the highest cumulative attention score.

$$\text{score}(j) = \sum_{i>j} A_{ij}, \qquad \text{retain top-}k \text{ by score}$$

Heavy hitters (high cumulative score) tend to be syntactically important tokens (verbs, subject nouns, sentence boundaries). H2O achieves $< 1\%$ accuracy degradation on downstream tasks while retaining only 20% of the KV cache.

Example — KV cache budget for LLaMA-3-70B, 32K context

Full KV cache per layer: $2 \times 32768 \times 8 \times 128 \times 2\,\text{B} = 134\,\text{MB}$. 80 layers: $10.7\,\text{GB}$.
H2O with $k = 4096$ retained tokens: $134/8 = 16.8\,\text{MB}$ per layer. 80 layers: $1.34\,\text{GB}$.
Fits in $40\,\text{GB}$ A100 alongside model weights ($\approx 35\,\text{GB}$).

Part 54 — Extended Supplement

Long-Context Scaling: Lost in the Middle, RULER Evaluation and Needle-in-a-Haystack

We derive the "Lost in the Middle" retrieval failure mathematically, analyse the RULER benchmark framework, and develop the Needle-in-a-Haystack test as an information-theoretic probing method.

54.7 Lost in the Middle: Attention Sink in Long Context

Liu et al. (2024) show that language models systematically fail to use information placed in the middle of long contexts, with the phenomenon worsening as context length grows.

Mathematical model. Consider a transformer processing a sequence of length n with the target information at position m. The attention score from output position n (the final token generating the answer) to position m:

$$A_{nm} = \frac{\exp(q_n^\top k_m / \sqrt{d_k})}{\sum_{j=1}^{n} \exp(q_n^\top k_j / \sqrt{d_k})}$$

As $n \to \infty$ and m is fixed (or $m/n \approx 0.5$), two effects emerge:

1. The partition function $Z_n = \sum_j \exp(q_n^\top k_j / \sqrt{d_k})$ grows with n (more denominator terms), causing $A_{nm} \to 0$.
2. RoPE's relative encoding means query q_n has highest inner product with recent keys (small $n - j$) due to well-trained frequency relationships.

Empirical finding. For multi-document QA, accuracy when the relevant document is in the first 20% of context: $\approx 70\%$. Last 20%: $\approx 65\%$. Middle 50%: $\approx 45\%$ — significantly worse, forming a U-shaped curve.

Mitigation: position interpolation with emphasis. Modified positional encodings that increase the "salience" of middle positions by applying a non-uniform scaling to the interpolation. Alternatively, instruction prompts that explicitly direct the model to "carefully read all provided context" shift the attention distribution (likely through the cross-attention mechanism in the final attention layers) to be more uniform.

54.8 RULER: Robust Context Understanding Evaluation

RULER (Hsieh et al. 2024) provides a systematic evaluation of long-context understanding across six task categories.

Needle retrieval (NIAH). A "needle" fact is inserted at position p/n within a haystack of irrelevant text. Accuracy is measured as a function of $(n, p/n)$. The effective context length L^* of a model is defined as the maximum n at which accuracy exceeds 95% for all needle positions $p/n \in [0.1, 0.9]$.

Formal test. Let h be a haystack sequence of length n and $f = (q, a)$ a fact (question-answer pair). Insert f at position m:

$$s = h[1:m] \oplus f \oplus h[m+1:n]$$

The model answers q given s. Accuracy $\text{Acc}(n, m) = P[\text{model}(s, q) = a]$.

The model passes NIAH at length n if $\min_{m \in [0.1n, 0.9n]} \text{Acc}(n, m) \geq 0.95$.

Multi-needle (MNIAH). Insert $K > 1$ facts at different positions and ask multi-hop questions requiring retrieval of all K facts. This tests the model's ability to maintain and combine multiple pieces of retrieved information. Difficulty scales as $O(K \log n)$: the model must attend to K positions in a length-n sequence.

Example — NIAH difficulty as a function of n

Assume attention score to the needle position m is:

$$A_{nm} \approx \frac{e^{s_{\text{needle}}}}{e^{s_{\text{needle}}} + (n-1)e^{s_{\text{background}}}}$$

where $s_{\text{needle}} > s_{\text{background}}$ is the signal-to-background ratio.
For the model to succeed with probability ≥ 0.95, we need $A_{nm} \geq \tau$ (some threshold for confident retrieval). Solving:

$$n_{\max} = \frac{e^{s_{\text{needle}}/\tau - s_{\text{background}}} + 1}{1} \approx \exp(s_{\text{needle}}/\tau - s_{\text{background}})$$

So the maximum successful context length grows *exponentially* in the signal-to-background ratio. A 1 nats improvement in attention signal doubles the effective context length.

54.9 Positional Generalisation: The ROPE Frequency Spectrum

The RoPE base $b = 10000$ sets the frequency spectrum of the positional encoding. The i-th dimension pair uses frequency $\theta_i = b^{-2i/d}$:

Dimension pair i	Frequency θ_i	Wavelength $\lambda_i = 2\pi/\theta_i$	Role
0	1.0	6.3 tokens	Short-range (consecutive)
$d/8 = 8$ (d=64)	0.25	25 tokens	Mid-range
$d/4 = 16$	0.01	628 tokens	Sentence-level
$3d/8 = 24$	3.2×10^{-4}	19,635 tokens	Document-level
$d/2 - 1 = 31$	10^{-4}	62,832 tokens	Cross-document

For a model trained on context length $L_{\text{train}} = 4096$, dimensions with $\lambda_i > L_{\text{train}}$ (i.e. the bottom $\approx 25\%$ of frequency pairs) have never completed a full revolution during training. These are the dimensions that extrapolation methods (YaRN, NTK) must handle carefully.

NTK interpolation formula. The NTK approach (bloc97, 2023) scales the base:

$$b' = b \cdot s^{d/(d-2)} \quad \text{for scale factor } s = L_{\text{test}}/L_{\text{train}}$$

This uniformly scales all wavelengths: $\lambda_i' = \lambda_i \cdot s$. The intuition: every frequency pair now sees the same effective fraction of its wavelength during training as the original model did. The formula $d/(d-2)$ arises from the requirement that the minimum wavelength $\lambda_0 = 2\pi/b^0 = 2\pi$ is preserved (high-frequency dimensions don't need scaling, only the low-frequency ones do — but uniform scaling is simpler and works empirically).

PART 55 — EXTENDED SUPPLEMENT

RWKV and Linear Recurrent Networks: Time-Mixing and Channel-Mixing

We derive the RWKV attention-free mechanism, analyse the time-mixing channel as a linear recurrence with learnable decay, and compare its expressiveness to Mamba's selective scan using the linear recurrence kernel perspective.

55.7 RWKV Architecture

RWKV (Peng et al. 2023) replaces attention with a linear recurrence while maintaining transformer-level performance on language modelling.

RWKV Time-Mixing

For position t and channel i, the time-mixing output is:

$$o_t = W_o \cdot \frac{\sum_{s \le t} e^{-(t-s-1)w + k_s} v_s + e^{u + k_t} v_t}{\sum_{s \le t} e^{-(t-s-1)w + k_s} + e^{u + k_t}}$$

where:

- $w \in \mathbb{R}^d$: learnable per-channel decay (positive, enforced by $w = \exp(\hat{w})$).
- $k_t = \mathbf{W}_k x_t \in \mathbb{R}^d$: token-dependent key.
- $v_t = \mathbf{W}_v x_t \in \mathbb{R}^d$: token-dependent value.
- $u \in \mathbb{R}^d$: learnable bias for the current token.

Recurrent form. Define numerator state a_t and denominator state b_t:

$$a_t = e^{-w} a_{t-1} + e^{k_{t-1}} v_{t-1}, \qquad b_t = e^{-w} b_{t-1} + e^{k_{t-1}}$$

Then:

$$o_t = W_o \cdot \frac{a_t + e^{u + k_t} v_t}{b_t + e^{u + k_t}}$$

This is a single-step recurrence with $O(d)$ state. Inference is $O(d)$ per token (no KV cache needed); training can be parallelised via parallel prefix scan.

Comparison to linear attention. The RWKV formula is equivalent to linear attention with a specific kernel $\phi(q_t, k_s) = e^{-(t-s)w + k_s}$, which factors as $e^{k_s} \cdot e^{-tw} \cdot e^{sw}$ — a product of a content-dependent term (e^{k_s}) and a position-dependent decay. The query plays no role in the kernel (it's absorbed into W_o), which is the key simplification enabling the stationary recurrence.

Channel mixing. The channel-mixing block applies:

$$x'_t = \text{sigmoid}(r_t) \odot (W'_v \cdot \max(m_t, 0)^2)$$

where $m_t = \mathbf{W}'_k(\mu x_t + (1-\mu) x_{t-1})$ is a linear interpolation of the current and previous token. The squared ReLU activation provides a form of gating without the multiplication cost of full SwiGLU.

EXAMPLE — RWKV state size vs. attention KV cache, $d = 4096$, $n = 32768$

Attention KV cache (32 heads, $d_k = 128$): $2 \times 32768 \times 32 \times 128 = 268M$ values.
RWKV state: 2 vectors per layer, size $d = 4096$ each: $2 \times 4096 = 8192$ values.

Ratio: $268M/8192 = 32{,}768\times$ — RWKV's memory cost is constant in sequence length.

RWKV-6: data-dependent receptance. The latest RWKV version introduces input-dependent w_t (like Mamba's Δ_t):

$$w_t = -\exp(\mathbf{W}_w x_t + b_w)$$

This allows the decay to be near-zero (long memory) or large (short memory) depending on the input, giving content-dependent selective memory similar to Mamba.

55.8 Griffin: Recurrent Blocks with Gated Linear Recurrence

Griffin (De et al. 2024) from Google interleaves local attention with gated linear recurrent (GLR) layers.

GLR Update

$$h_t = \Lambda_t \odot h_{t-1} + (1 - \Lambda_t) \odot (W_x x_t)$$
$$\Lambda_t = \sigma(\mathbf{W}_\Lambda x_t + b_\Lambda) \in (0,1)^d$$

The gate Λ_t interpolates between preserving the past state (when $\Lambda_t \approx 1$) and replacing it with the current input (when $\Lambda_t \approx 0$). This is a learnable, input-dependent version of the EMA (exponentially weighted moving average) filter $h_t = \alpha h_{t-1} + (1-\alpha)x_t$, but with α varying per timestep and per channel.

Relationship to GRU. The GLR update is a simplified GRU (Gated Recurrent Unit) where the reset gate is removed. The full GRU has:

$$z_t = \sigma(\mathbf{W}_z[h_{t-1}; x_t]), \quad r_t = \sigma(\mathbf{W}_r[h_{t-1}; x_t]), \quad h_t = (1-z_t)\odot h_{t-1} + z_t \odot \tanh(\mathbf{W}[r_t \odot h_{t-1}; x_t])$$

The GLR simplifies by (a) not using h_{t-1} in the gate computation, (b) removing the reset gate, (c) using a linear update instead of tanh. These simplifications make the GLR amenable to parallel prefix scan (since the update is linear in h_{t-1}) while retaining gating expressiveness.

PART 56 — EXTENDED SUPPLEMENT

Attention as Kernel Smoothing and the RKHS Perspective

We establish the Nadaraya-Watson kernel regression interpretation of attention, derive the associated RKHS function class, and analyse the universal approximation property of transformers through this lens.

56.3 Attention as Nadaraya-Watson Estimation

Nadaraya-Watson Estimator

Given observations $\{(k_i, v_i)\}_{i=1}^n$, the Nadaraya-Watson estimator with kernel $\mathcal{K}$ produces:

$$\hat{f}(q) = \frac{\sum_{i=1}^n \mathcal{K}(q, k_i) v_i}{\sum_{i=1}^n \mathcal{K}(q, k_i)}$$

Standard softmax attention is exactly Nadaraya-Watson with the exponential dot-product kernel $\mathcal{K}(q, k) = \exp(q^\top k/\sqrt{d})$. This interpretation reveals several properties:

Consistency. If keys $\{k_i\}$ are i.i.d. samples from a distribution p_k, and the function $f(k) = v$ satisfies Lipschitz continuity, then as $n \to \infty$:

$$\hat{f}(q) \xrightarrow{a.s.} \frac{\int \mathcal{K}(q, k) f(k) p_k(k)\, dk}{\int \mathcal{K}(q, k) p_k(k)\, dk} = \mathbb{E}_{k \sim p_{\mathcal{K}(\cdot|q)}}[f(k)]$$

where $p_{\mathcal{K}(k|q) \propto \mathcal{K}(q,k) p_k(k)}$. Attention computes a conditional expectation under the kernel-weighted distribution.

Bias-variance tradeoff. The temperature τ in attention controls the bandwidth of the kernel. For $\mathcal{K}(q, k) = \exp(q^\top k/\tau)$:

- $\tau \to 0$: kernel becomes a Dirac delta at the nearest neighbour. Zero bias (exact retrieval) but high variance (sensitive to small query perturbations).
- $\tau \to \infty$: uniform kernel. High bias (averages all values regardless of relevance) but zero variance.
- Optimal τ: balances these, analogous to kernel bandwidth selection in nonparametric statistics.

Attention as kernel regression. The learned query and key projections $q = \mathbf{W}_Q x$, $k_i = \mathbf{W}_K x_i$ define a feature space where attention performs Nadaraya-Watson regression. The attention mechanism thus learns a kernel implicitly through the projections $\mathbf{W}_Q, \mathbf{W}_K$.

56.4 Universal Approximation for Transformers

Theorem (Yun et al. 2020)

Transformers with $O(\log n)$ layers, $O(d)$ width, and positional encoding are universal approximators of sequence-to-sequence functions in L^1. Specifically, for any $\epsilon > 0$ and continuous $f : [0, 1]^{n \times d} \to [0, 1]^{n \times d}$, there exists a transformer T such that $\|T - f\|_{L^1} \leq \epsilon$.

Proof sketch. The key steps are:

1. *Contextualize tokens*: show that attention with a hard (one-hot) attention pattern can copy any token to any position.
2. *Apply arbitrary functions*: show that FFN layers can approximate any pointwise function.
3. *Compose*: combine these to approximate arbitrary sequence-to-sequence maps.

The attention head with keys $k_j = \text{onehot}(j)$ and queries $q_i = \text{onehot}(j_0)$ for target position j_0 achieves hard attention: the output at position i is exactly v_{j_0}. This implements a "copy" operation that moves any value from any position to any other.

The FFN layer implements pointwise transformations: with ReLU activation and width $O(d^2)$, an FFN can approximate any continuous function on $\mathbb{R}^d$ to within ϵ (classical universal approximation).

Limitations. The universal approximation result is existential (some transformer exists) not constructive (gradient descent finds it). The required depth $O(\log n)$ for n tokens means transformers need depth proportional to the log of the sequence length for full universality — consistent with the chain-of-thought results of Part 59.

PART 57 — EXTENDED SUPPLEMENT

Inference-Time Compute Scaling: Process Reward Models and Best-of-N

We derive the sample efficiency of Best-of-N sampling under a process reward model, formalise the verifier-prover architecture, and analyse the compute-optimal allocation between training and inference for reasoning tasks.

57.7 Best-of-N Sampling with Verifiers

Let p be the per-attempt probability that the model produces a correct solution. With N independent samples and a perfect verifier:

$$P(\text{at least one correct}) = 1 - (1-p)^N$$

For small p: $P \approx 1 - e^{-Np}$. To achieve $P \geq 1 - \delta$: $N \geq \ln(1/\delta)/p$.

Compute equivalence. A model with per-attempt accuracy p using N samples is equivalent (in success probability) to a model with accuracy p' using 1 sample, where:

$$p' = 1 - (1-p)^N \approx Np \text{ for small } p$$

So N samples multiplicatively amplifies the effective accuracy. Crucially, inference compute scales as $O(N)$ while training compute would need to scale as $O(1/p)$ to achieve the same per-attempt accuracy improvement. For small p, inference is more efficient.

57.8 Process Reward Models (PRMs)

A Outcome Reward Model (ORM) scores complete solutions. A Process Reward Model (PRM) scores individual reasoning steps $z_1, z_2, \ldots, z_T$:

$$r_{\text{PRM}}(z_{1:T}) = \prod_{t=1}^{T} P_{\text{step}}(\text{step } t \text{ is correct} | z_{1:t})$$

Using the PRM, we can perform *step-level* beam search:

1. Maintain a beam of B partial solutions.
2. At each step, generate k continuations for each beam element.
3. Score each continuation with the PRM.
4. Retain the top B partial solutions by PRM score.

Compute cost. T steps with beam width B and branching factor k: $T \times B \times k$ model calls for generation + $T \times B \times k$ PRM calls. Total: $O(TBk)$ vs. $O(TN)$ for Best-of-N (where $N = (Bk)^T$ to cover the same tree).

Why PRMs outperform ORMs for long reasoning. For a T-step solution where each step has accuracy p:

- ORM Best-of-N: needs $N \approx p^{-T}$ samples to find a correct solution.
- PRM beam search width B: needs $B \approx 1/p$ to maintain a correct path.

PRM requires $O(BT) = O(T/p)$ total compute vs. ORM's $O(p^{-T})$. For $T = 10$, $p = 0.7$: PRM: $O(14)$ vs. ORM: $O(0.7^{-10}) \approx 28$ — already more efficient, and the gap grows exponentially with T.

Example — Inference scaling law for MATH (competition mathematics)

Per-step model accuracy: $p_{\text{step}} = 0.85$ (from PRM training on process labels). Solution length: $T = 6$ steps on average. Per-solution accuracy: $p_{\text{sol}} = p_{\text{step}}^{T} = 0.85^{6} \approx 0.38$. Best-of-N to achieve 95% success: $N = \ln(0.05)/\ln(0.62) \approx 6$ attempts. PRM beam width $B = 4$: covers $4 \times 0.85/0.15 \approx 23$ expected solution paths per step. PRM solve rate at $B = 4$: $\approx 1 - (1 - 0.85)^{4} \approx 0.998$ per step — near-perfect.

57.9 Reasoning Tokens and the Extended Chinchilla Model

Let C_{inf} be inference compute allocated per query as a budget of reasoning tokens T_{inf}. The query-level loss:

$$L(N, D, T_{\text{inf}}) = E + \frac{A}{N^{\alpha}} + \frac{B}{D^{\beta}} + \frac{F}{T_{\text{inf}}^{\gamma}}$$

where $\gamma \approx 0.5$ (Snell et al. 2024). The three-way optimisation subject to $C_{\text{total}} = 6ND + n_{\text{queries}} T_{\text{inf}} N_{\text{inf}}$ (where N_{inf} is the inference model size, assumed equal to N for simplicity):

At the optimum, the marginal loss reduction per FLOP must be equal across all three terms:

$$\frac{A\alpha}{N^{\alpha+1}} \cdot \frac{C}{6D} = \frac{B\beta}{D^{\beta+1}} \cdot \frac{C}{6N} = \frac{F\gamma}{T_{\text{inf}}^{\gamma+1}} \cdot \frac{C}{n_{\text{queries}} N}$$

With $\alpha = 0.34$, $\beta = 0.28$, $\gamma = 0.5$: the inference term has the steepest exponent, meaning an additional FLOP allocated to inference gives the highest marginal return. This is the theoretical basis for "inference-time compute scaling" being complementary to training scaling.

PART 58 — EXTENDED SUPPLEMENT

Predictability of Emergence: Smooth Scaling of Underlying Metrics

We derive the Schaeffer et al. result that many emergent abilities are artefacts of discontinuous metric choice, derive the continuous underlying metrics, and analyse the phase transition in modular arithmetic via Fourier analysis of neural network weights.

58.6 The Metric-Artefact Hypothesis

Schaeffer et al. (2023) argue that emergent abilities are artefacts of discontinuous evaluation metrics, not genuine phase transitions in the model's capabilities.

Formal statement. Let X be a continuous underlying capability metric (e.g. logit of the correct answer). Define a discontinuous evaluation metric $Y = \mathbf{1}[X \geq \tau]$ (accuracy-style). If X scales smoothly with compute C:

$$X(C) = a - be^{-cC}$$

then $Y(C) = \mathbf{1}[X(C) \geq \tau]$ transitions sharply at $C^* = -\ln((a-\tau)/b)/c$. The apparent emergence is a consequence of the thresholding, not a discontinuity in X.

Rescaling reveals continuity. The continuous metric X (e.g. cross-entropy per token on the task, or the log probability assigned to the correct answer) scales smoothly. Only after applying the threshold ("what fraction of the time does the model give the exact correct answer?") does the apparent discontinuity appear.

Counter-examples (genuine emergence). Some capabilities resist the metric-artefact explanation. Multi-step arithmetic circuits, where the model must execute a full algorithm (carry propagation, etc.) correctly in all steps, show genuine phase transitions. The reason: algorithm execution is compositional — getting k out of T steps right is near-chance on the overall task unless $k = T$. Smooth improvement in per-step accuracy maps discontinuously to task accuracy. This is structural emergence, not metric artefact.

58.7 Circuit Formation and Grokking Phase Diagram

The grokking phenomenon (sudden generalisation after prolonged memorisation) can be understood via a two-circuit competition model.

Two-circuit model. Decompose model weights as $W = W_M + W_G$ where:

- W_M: memorisation circuit (lookup table mapping training inputs to outputs).
- W_G: generalisation circuit (algorithmic solution, e.g. Fourier-based for modular arithmetic).

Both circuits develop simultaneously. Loss on training data:

$$L_{\text{train}} = \ell(W_M + W_G, D_{\text{train}}) \approx 0 \quad \text{(both circuits memorise)}$$

Loss on test data:

$$L_{\text{test}} = \ell(W_M + W_G, D_{\text{test}}) \approx \ell(W_G, D_{\text{test}}) + \text{cross terms}$$

since W_M memorises training data, not test data.

Weight decay dynamics. With L2 regularisation $\lambda\|W\|^2$:

$$\frac{d}{dt}\|W_M\|^2 = -2\lambda\|W_M\|^2 - 2\eta\langle\nabla_{W_M}L, W_M\rangle$$

Once $L_{\text{train}} \approx 0$, $\nabla_{W_M}L \approx 0$ and:

$$\|W_M(t)\|^2 \approx \|W_M(0)\|^2 e^{-2\lambda t} \to 0$$

The memorisation circuit decays exponentially. The generalisation circuit W_G has smaller norm (efficient algorithm) and receives gradient signal from test-like data patterns. When $\|W_M\| \ll \|W_G\|$, the test loss drops — this is grokking.

Phase diagram. The grokking "phase" depends on two parameters: $(1/\lambda, |\mathcal{D}_{\text{train}}|)$:

- Small λ, large dataset: quick generalisation (strong regularisation kills memorisation fast; large dataset helps generalisation circuit form quickly).
- Large λ, small dataset: quick memorisation, no generalisation (all gradient goes to small memorisation circuit).
- Intermediate: grokking regime — long memorisation phase, then sudden generalisation.

PART 60 — EXTENDED SUPPLEMENT

ColBERT: Late Interaction and Multi-Vector Retrieval

We derive the ColBERT MaxSim scoring function, analyse its relationship to single-vector DPR, and prove the expressive advantage of late interaction for queries with multiple sub-intents.

60.4 ColBERT Late Interaction

DPR computes similarity as a single dot product between query and passage embeddings. ColBERT (Khattab & Zaharia, 2020) uses late interaction: all token-level query and passage embeddings interact.

ColBERT MaxSim Scoring

Let $E_Q(q) = [\mathbf{q}_1, \ldots, \mathbf{q}_m] \in \mathbb{R}^{m \times d}$ be per-token query embeddings and $E_P(p) = [\mathbf{p}_1, \ldots, \mathbf{p}_k] \in \mathbb{R}^{k \times d}$ passage embeddings. The MaxSim score:

$$\text{ColBERT}(q, p) = \sum_{i=1}^{m} \max_{j \in [k]} (\mathbf{q}_i^\top \mathbf{p}_j)$$

Each query token matches its most similar passage token; scores are summed.

Relation to DPR. DPR's [CLS] dot product is: $\mathbf{q}_{\text{CLS}}^\top \mathbf{p}_{\text{CLS}}$, a single interaction. ColBERT has m interactions, each finding the best passage token for each query token.

Expressive advantage. For a multi-intent query ("transformer architecture efficiency"), ColBERT's query tokens independently match: $\mathbf{q}_{\text{transformer}}$ matches passage tokens about transformers, $\mathbf{q}_{\text{efficiency}}$ matches tokens about efficiency. DPR's single [CLS] must encode both intents in one vector, limiting its discriminability.

MaxSim as a special kernel. Define the MaxSim kernel: $\mathcal{K}_{\text{MaxSim}}(q, p) = \sum_i \max_j \mathbf{q}_i^\top \mathbf{p}_j$. This is not positive definite (no RKHS exists), but it satisfies: $\mathcal{K}_{\text{MaxSim}}(q, q) \geq \mathcal{K}_{\text{MaxSim}}(q, p)$ when p is less relevant — a necessary condition for use in ranking.

Precomputation. At index time, all passage embeddings are precomputed and stored. Retrieval requires computing inner products between m query vectors and all passage vectors — MIPS in $\mathbb{R}^{m \times d}$. FAISS PLAID (a GPU-optimised ColBERT retrieval engine) accelerates this by (1) approximate centroid-based filtering (reducing candidates from $|\mathcal{C}|$ to K) and (2) exact MaxSim computation on candidates. Cost: $O(m|\mathcal{C}|/r + mKd)$ for r centroids.

Part 61 — Extended Supplement

Attention Head Typology: Induction Heads, Copy Suppression and Virtual Weights

We derive the induction head mechanism mathematically, prove that induction heads implement in-context learning, formalise copy suppression heads, and develop the virtual weight framework for composing attention heads.

61.7 Induction Heads

An induction head (Olsson et al. 2022) is an attention head that implements "if you've seen [A][B], predict B to follow A". This is the minimal unit of in-context learning.

Formal definition. An induction head in layer ℓ computes:

$$h^{(\ell)}_{\text{ind}}(i) = \sum_j A_{ij} \mathbf{W}_V x^{(\ell)}_j$$

where A_{ij} is largest for the j such that $x^{(\ell)}_{j-1} \approx x^{(\ell)}_i$ (the token *before* j matches the current token i).

Implementation via two-layer composition. Layer 1 contains a "previous token head": $h^{(1)}_{\text{prev}}(j) = \mathbf{W}^{(1)}_V x^{(0)}_{j-1}$ (attends to the previous token, copying it into the residual stream at position j).

Layer 2 contains the induction head, which uses the key computed from the layer-1 output (which includes the previous-token information):

$$k^{(2)}_j = \mathbf{W}^{(2)}_K (x^{(0)}_j + h^{(1)}_{\text{prev}}(j))$$

If $\mathbf{W}^{(2)}_K h^{(1)}_{\text{prev}}(j) \approx \mathbf{W}^{(2)}_Q x^{(0)}_i$ when $x^{(0)}_{j-1} \approx x^{(0)}_i$, then the induction head attends to the token immediately following any past occurrence of the current token.

Mathematical condition. The head is a perfect induction head when: $\mathbf{W}^{(2)}_Q \mathbf{W}^{(2)\top}_K \approx \mathbf{W}^{(1)}_K \mathbf{W}^{(1)\top}_V$ (the query-key product in layer 2 equals the key-value product in layer 1). This is the "QK circuit" composition across layers.

Example — Induction head in GPT-2 small, layer 5 head 1

Pattern: for token A at position i, the head attends to all positions j where token $x_{j-1} = A$. Specifically, for the sequence "...the cat sat on the mat...the cat", position $i =$ "cat" (second occurrence) attends strongly to position $j =$ "sat" (follows first "cat").
This implements: "the token after the last occurrence of [current token] is likely to follow again." For n-shot in-context learning with n examples, n induction head activations accumulate evidence for the pattern.

61.8 Virtual Weights and Composition

Virtual Weight Matrix

When a query-key circuit h_ℓ writes to the residual stream and a later head $h_{\ell'}$ reads from it, the effective QK matrix of $h_{\ell'}$ as seen from the input is the composition:

$$W_{QK}^{\text{virt}} = \mathbf{W}_Q^{(\ell')} \cdot \mathbf{W}_V^{(\ell)} \cdot \mathbf{W}_K^{(\ell')\top}$$

This is a rank-d_k matrix in the original $d_{\text{model}} \times d_{\text{model}}$ space, describing which input-space features $h_{\ell'}$ uses to determine its attention.

Information-theoretic interpretation. The virtual weight W_{QK}^{virt} has at most $\min(d_k, d_v)$ non-trivial singular values. Since $d_k \ll d_{\text{model}}$ (typically $d_k = d_{\text{model}}/H$), attention heads operate in a low-dimensional information bottleneck. The SVD of W_{QK}^{virt} reveals which input features the circuit is sensitive to.

Copy suppression. Certain heads have negative effective OV circuits: $\mathbf{W}_O\mathbf{W}_V \approx -\lambda\mathbf{I}$ (negative identity). These heads suppress the direct copy of tokens into the output, preventing "just copy the input" degenerate solutions. Mechanistically, copy suppression heads write directions *opposite* to the logit of the attended token, decreasing its probability.

Part 62 — Extended Supplement

GRPO: Group Relative Policy Optimisation and Reward Scaling

We derive Group Relative Policy Optimisation (GRPO) as used in DeepSeek-R1, analyse the advantages estimator for reasoning tasks, and compare GRPO to PPO in terms of variance reduction and memory efficiency.

62.7 GRPO: Group Relative Policy Optimisation

PPO requires a value function (critic) that estimates $V(s_t)$, adding significant memory cost ($\approx 2\times$ model parameters). GRPO (Shao et al. 2024) eliminates the critic by computing advantages relative to a group of outputs.

GRPO Objective

For prompt x, sample G outputs $\{y_1, \dots, y_G\} \sim \pi_\theta(\cdot|x)$. Compute rewards $\{r_1, \dots, r_G\}$. Normalised advantage for output i:

$$\hat{A}_i = \frac{r_i - \text{mean}(\{r_j\})}{\text{std}(\{r_j\})}$$

GRPO loss:

$$\mathcal{L}_{\text{GRPO}} = -\frac{1}{G}\sum_{i=1}^{G} \min\left(\frac{\pi_\theta(y_i|x)}{\pi_{\text{old}}(y_i|x)}\hat{A}_i,\ \text{clip}\left(\frac{\pi_\theta}{\pi_{\text{old}}}, 1-\epsilon, 1+\epsilon\right)\hat{A}_i \right) + \beta D_{KL}(\pi_\theta \| \pi_{\text{ref}})$$

Why GRPO works without a critic. The advantage $\hat{A}_i$ measures how much better output i is compared to the group average. For a well-chosen reward function (e.g. binary correctness for math problems), the variance of rewards within a group provides a natural normalisation.

Variance analysis. In PPO, the advantage $A_t = r_t + \gamma V(s_{t+1}) - V(s_t)$ has variance dominated by the critic's estimation error. In GRPO, the advantage variance is:

$$\text{Var}(\hat{A}_i) = 1 \quad \text{(by construction, after normalisation)}$$

For binary rewards ($r_i \in \{0, 1\}$), the pre-normalisation variance is $p(1-p)$ where p is the per-group accuracy. Normalised advantage has unit variance regardless of p.

Memory efficiency. PPO: 4 models in memory (policy, reference, critic, reward). GRPO: 2 models (policy, reference). For a 70B model, GRPO saves ≈ 140 GB GPU memory during training — critical for fitting training on available hardware.

Example — GRPO reward signals for mathematical reasoning

Group size $G = 8$ outputs per prompt. Reward: $r_i = 1$ if answer correct, else 0. Suppose 3 of 8 outputs are correct.
Mean reward: $3/8 = 0.375$. Std: $\sqrt{(3\times(1-0.375)^2 + 5\times(0-0.375)^2)/8} = \sqrt{(3\times 0.391 + 5\times 0.141)/8} = \sqrt{(1.172+0.703)/8} = \sqrt{0.234} = 0.484$.
Correct outputs: $\hat{A} = (1-0.375)/0.484 = 1.29$ (positive, encouraged). Incorrect outputs: $\hat{A} = (0-0.375)/0.484 = -0.775$ (negative, discouraged).
The policy is pushed to increase probability of the 3 correct formats and decrease the 5 incorrect ones.

62.8 Reward Shaping for Mathematical Reasoning

DeepSeek-R1 uses two types of rewards:

Format reward. Binary reward for producing the expected output format (e.g. wrapping the final answer in $\langle$`answer`$\rangle$ tags):

$$r_{\text{format}} = \mathbf{1}[\text{response follows format}]$$

Accuracy reward. Binary reward for mathematical correctness:

$$r_{\text{accuracy}} = \mathbf{1}[\text{answer is numerically correct}]$$

Combined reward: $r = r_{\text{format}} + r_{\text{accuracy}}$.

Why not use a learned reward model? For math and coding tasks, correctness can be verified automatically (symbolic math evaluation, code execution). Learned reward models are expensive and subject to reward hacking. Verifiable rewards sidestep both problems.

Curriculum. Difficulty of training problems is gradually increased. Early training uses problems where $p_{\text{solve}} \approx 0.5$, providing strong gradient signal. If $p_{\text{solve}} \approx 1$: all rewards equal, no gradient. If $p_{\text{solve}} \approx 0$: no correct samples in group, no positive gradient. Optimal $p \approx 0.5$ maximises the expected number of informative contrasts per group.

PART 66 — EXTENDED SUPPLEMENT

Adapter Tuning, Prefix Tuning and Prompt Tuning: A Unified View

We derive the mathematical relationship between adapter tuning, prefix tuning, and LoRA, showing they are special cases of a unified low-rank update framework, and analyse the expressiveness tradeoffs via SVD decomposition.

66.5 Adapter Tuning

Adapters (Houlsby et al. 2019) insert small trainable bottleneck modules between frozen transformer layers:

$$h' = W_{\text{up}}(\text{ReLU}(W_{\text{down}}h)) + h$$

with $W_{\text{down}} \in \mathbb{R}^{r\times d}$, $W_{\text{up}} \in \mathbb{R}^{d\times r}$, $r \ll d$.

Relationship to LoRA. The adapter update on the hidden state:

$$\Delta h = W_{\text{up}}W_{\text{down}}h = (W_{\text{up}}W_{\text{down}})h$$

This is equivalent to adding $\Delta W = W_{\text{up}}W_{\text{down}} \in \mathbb{R}^{d\times d}$ (rank r) to the weight matrix of the following layer. LoRA makes this explicit by parameterising $\Delta W = BA$ directly. The difference: adapters apply after a nonlinearity (ReLU), while LoRA applies to the linear weights before the nonlinearity.

Inference overhead. Adapters require a separate forward pass through the bottleneck at every layer. For $r/d = 1/64$ bottleneck ratio and L layers: overhead $\approx 2Lr/d = 2L/64 \approx 3\%$ of total FLOPs. LoRA can be merged into the base weights ($W' = W_0 + BA$) with zero inference overhead.

66.6 Prefix Tuning

Li & Liang (2021) prepend k learnable "prefix" tokens $P_k = (p_1, \dots, p_k)$ to the key and value sequences of every attention head:

$$\text{Attn}(q_i, [K; P_K], [V; P_V]) = \text{softmax}\left(\frac{q_i[k_1; \dots; k_n; p_1; \dots; p_k]^\top}{\sqrt{d_k}}\right)[v_1; \dots; v_n; p_1; \dots; p_k]$$

Virtual token perspective. The prefix tokens act as additional context that all real tokens can attend to. From the attention head's perspective, it has $n+k$ keys and values, where the first n are data-dependent and the last k are learned parameters.

Relationship to LoRA. Let $\tilde{K}_{n+k} = [\mathbf{W}_K X; P_K]$ be the augmented key matrix. The attention output for real token i:

$$o_i = \underbrace{\alpha_{\text{real}} \sum_{j=1}^{n} A_{ij} v_j}_{\text{standard attention}} + \underbrace{\alpha_{\text{prefix}} \sum_{k=1}^{K} A_{i,n+k} p_k^V}_{\text{prefix contribution}}$$

The prefix attention term is equivalent to adding $\sum_k A_{i,n+k} p_k^V$ to the output — a content-dependent weighted combination of learnable vectors. This is different from LoRA in that

the "weights" (the A_{ik} values) are computed dynamically based on the query, while LoRA's ΔW is a fixed matrix.

Unified framework. All PEFT (Parameter-Efficient Fine-Tuning) methods can be written as:

$$W_{\text{eff}} = W_0 + \Delta(W_0, x, \theta_{\text{PEFT}})$$

where Δ is:

- LoRA: $\Delta = BA$ (independent of input x, low-rank).
- Adapter: $\Delta = W_{\text{up}}\sigma(W_{\text{down}}x)x^{-1}$ (pseudo-inverse, nonlinear, input-dependent).
- Prefix: Δ depends on x through attention weights (highly nonlinear).
- IA3 (Liu et al. 2022): $\Delta = \text{diag}(l)W_0$ (scaling, multiplicative, rank-full but sparse).

Part 68 — Extended Supplement

Stochastic Differential Equations for Generative Modelling

We present Song et al.'s (2021) unified SDE framework for diffusion models, derive the VE-SDE and VP-SDE formulations, and prove the equivalence between the probability flow ODE and the reverse-time SDE.

68.4 The SDE Framework

Song et al. (2021) unify all continuous diffusion processes under the SDE framework:

Forward SDE

$$dx = f(x,t)\,dt + g(t)\,dW$$

where $f : \mathbb{R}^d \times [0,T] \to \mathbb{R}^d$ is the drift coefficient, $g : [0,T] \to \mathbb{R}$ is the diffusion coefficient, and W is a standard Wiener process.

VE-SDE (Variance Exploding). Used in NCSN:

$$dx = g(t)\,dW, \qquad g(t) = \sqrt{\frac{d\sigma^2}{dt}}$$

No drift; the variance grows from σ_{min}^2 to σ_{max}^2. Marginal distribution: $p_t(x) = \int p_0(y)\mathcal{N}(x; y, \sigma^2(t))\,dy = p_0 * \mathcal{N}(0, \sigma^2(t))$.

VP-SDE (Variance Preserving). Used in DDPM:

$$dx = -\frac{\beta(t)}{2}x\,dt + \sqrt{\beta(t)}\,dW$$

The ODE preserves a variance-1 Gaussian marginal: if $p_0 = \mathcal{N}(0, \mathbf{I})$ then $p_t = \mathcal{N}(0, \mathbf{I})$ for all t. For data distribution $p_0 \neq \mathcal{N}(0, \mathbf{I})$: $p_t \to \mathcal{N}(0, \mathbf{I})$ as $t \to T$.

Reverse-time SDE. Anderson (1982) derived the reverse-time SDE:

$$dx = \left[f(x,t) - g(t)^2 \nabla_x \log p_t(x)\right] dt + g(t)\,d\bar{W}$$

where $\bar{W}$ is a Wiener process running backwards in time. To simulate the reverse process, we need $\nabla_x \log p_t(x)$ — the score function. The DDPM noise predictor estimates this: $s_\theta(x_t, t) = -\epsilon_\theta(x_t, t)/\sigma_t$.

Probability flow ODE. Every SDE has a corresponding ODE with the same marginal densities $\{p_t\}_t$:

$$dx = \left[f(x,t) - \frac{1}{2}g(t)^2 \nabla_x \log p_t(x)\right] dt$$

This ODE is deterministic (no dW term) and produces the same marginal at each time step as the stochastic process. The difference: the SDE explores the full distribution at each t, while the ODE follows deterministic trajectories.

Example — VP-SDE score function at $t = 0.5$, $\beta(t) = 10^{-4} + (0.02 - 10^{-4})t$

$\bar{\alpha}_t = \exp(-\int_0^t \beta(s)ds)$. At $t = 0.5$: $\int_0^{0.5} \beta(s)ds = 10^{-4}(0.5) + (0.02 - 10^{-4})(0.5)^2/2 = 5 \times 10^{-5} + 0.00499 \approx 0.005$. $\bar{\alpha}_{0.5} = e^{-0.005} \approx 0.995$.
Score: $\nabla_x \log p_{0.5}(x) \approx -\frac{x - \sqrt{\bar{\alpha}_{0.5}}x_0}{\sigma_{0.5}^2} = -\frac{\epsilon}{\sqrt{1-\bar{\alpha}_{0.5}}} \approx -\frac{\epsilon}{0.0707}$

where $x = \sqrt{0.995}x_0 + 0.0707\epsilon$. At early time ($t = 0.5$ out of $T = 1$), $\bar{\alpha}$ is close to 1 and the score is large (strong signal about x_0).

68.5 Consistency Between SDE Formulations

All three formulations — DDPM (discrete Markov chain), score-based SDE, and flow matching — are equivalent in the continuous-time limit.

Theorem. For the VP-SDE with $\beta(t) = \bar{\beta} + (\bar{\beta}_T - \bar{\beta}) \cdot t/T$, discretised with step size $\Delta t = 1/T$:

$$x_{t-\Delta t} = x_t + \frac{\beta_t}{2} x_t \Delta t + \beta_t (\nabla_x \log p_t(x_t)) \Delta t + \sqrt{\beta_t \Delta t}\epsilon$$

This recovers the DDPM reverse step with $\mu_\theta(x_t, t) = x_t + \frac{\beta_t}{2}[x_t + 2\nabla_x \log p_t(x_t)]$ and noise $\sigma = \sqrt{\beta_t \Delta t}$.

Flow matching as a deterministic SDE. Flow matching with the linear path $\mathbf{z}_t = (1-t)x_0 + tx_1$ corresponds to the ODE:

$$\frac{dx}{dt} = x_1 - x_0 = E_{x_0, x_1 \sim \pi}[x_1 - x_0 | x_t]$$

This is the deterministic limit ($g(t) = 0$) of an SDE with drift equal to the optimal transport velocity field. Under the OT coupling $\pi = \pi^*$ (the transport plan), the velocity field is constant along trajectories, making the ODE trivially integrable with a single step.

Appendix B: Notation Guide for Volume V

Attention and transformers

n	Sequence length
d	Model dimension
H	Number of attention heads
$d_k = d/H$	Key/query dimension per head
w	Sliding window size (SWA)
G	Number of GQA groups; also global tokens
d_c	MLA KV latent dimension
b	RoPE base frequency ($b = 10000$)
$\theta_i = b^{-2i/d}$	RoPE rotation frequency for pair i
$\lambda_i = 2\pi/\theta_i$	RoPE wavelength for pair i
m_h	ALiBi slope for head h
s	YaRN/NTK scale factor $= L_{\text{test}}/L_{\text{train}}$

State space models

N	SSM state dimension
$\mathbf{A} \in \mathbb{R}^{N\times N}$	State transition matrix (continuous)
$\bar{\mathbf{A}} = e^{\Delta\mathbf{A}}$	Discretised state matrix
Δ_k	Input-dependent time step (Mamba)
$c_n(t)$	HiPPO Legendre polynomial coefficients
$a_t \in (0,1)^N$	Mamba/SSD per-step decay
$\mathbf{L}_{ts} = \prod_{r=s+1}^{t} a_r$	SSD cumulative decay
DPLR	Diagonal Plus Low-Rank SSM parameterisation

Scaling and emergence

$L(N, D)$	Language model loss as function of size and tokens
E	Irreducible entropy floor
$\alpha \approx 0.34$	Chinchilla model-size exponent
$\beta \approx 0.28$	Chinchilla data-size exponent
$C = 6ND$	Training compute in FLOPs
N^*, D^*	Chinchilla-optimal model size and token count
T_{inf}	Inference-time reasoning tokens
$\gamma \approx 0.5$	Inference-time scaling exponent

Alignment and fine-tuning

$r_\phi(x, y)$	Reward model score
$\pi_\theta, \pi_{\text{ref}}$	Learned and reference policies
β	KL penalty coefficient in RLHF
$D_{\text{KL}}(\pi_\theta \Vert \pi_{\text{ref}})$	Policy divergence from reference
W_0	Frozen pretrained weight matrix
$\Delta W = BA$	LoRA low-rank update ($r \ll d$)
d_{90}	Intrinsic dimension (90% performance at rank d_{90})
G	GRPO group size (number of sampled outputs)
$\widehat{A}_i$	Normalised group relative advantage

Interpretability

$\mathbf{W}_E \in \mathbb{R}^{V \times d}$	Token embedding matrix
$\mathbf{W}_U \in \mathbb{R}^{d \times V}$	Unembedding (logit) matrix
$\mathrm{OV} = \mathbf{W}_O \mathbf{W}_V$	Output-value circuit
$\mathrm{QK} = \mathbf{W}_Q^\top \mathbf{W}_K$	Query-key circuit
P_j	Polysemanticity index of neuron j
$\mathbf{D} \in \mathbb{R}^{d \times m}$	SAE decoder dictionary
K	TopK SAE active feature count

PART 70

Speculative Decoding and Inference Acceleration

Speculative decoding uses a small draft model to generate candidate tokens that a large target model verifies in parallel. We derive the acceptance rate formula, prove losslessness, and analyse the latency-speedup tradeoff.

70.1 The Speculative Decoding Protocol

Standard autoregressive decoding generates tokens sequentially: each forward pass produces one token. The bottleneck is memory bandwidth (loading model weights), not arithmetic throughput. Speculative decoding (Leviathan et al. 2023) batches multiple token verifications in a single forward pass.

Speculative Decoding Algorithm

Given target model p and draft model q (much smaller):

1. Draft model generates K tokens autoregressively: $\tilde{x}_1, \ldots, \tilde{x}_K$.
2. Target model processes all K draft tokens in *one* parallel forward pass, computing $p(\cdot|x_{1:t+k})$ for $k = 0, \ldots, K-1$.
3. For each position $k = 1, \ldots, K$, accept $\tilde{x}_k$ with probability:

$$\alpha_k = \min\left(1, \frac{p(\tilde{x}_k|x_{1:t+k-1})}{q(\tilde{x}_k|x_{1:t+k-1})}\right)$$

4. On rejection at position k, sample the next token from the adjusted distribution:

$$x \sim \text{norm}(\max(0, p(\cdot|x_{1:t+k-1}) - q(\cdot|x_{1:t+k-1})))$$

5. Accept one token beyond the last accepted draft token unconditionally.

Losslessness. The key property: the tokens produced by speculative decoding are *identically distributed* to those from the target model p, regardless of the quality of the draft model q.

Proof. For a single token at position t, we show that the marginal distribution of the output x_t is $p(x_t|x_{1:t-1})$. Consider token x:

$$P(\text{output} = x) = P(\text{accept } x) + P(\text{reject, sample } x \text{ from adjusted})$$

Case 1: Draft $= x$. Output is x iff accepted:

$$P_{\text{case1}} = q(x) \cdot \min(1, p(x)/q(x)) = \min(q(x), p(x))$$

Case 2: Draft $= y \neq x$, rejected, resample x. Prob of rejection:

$$P_{\text{reject}} = \sum_y q(y)(1 - \min(1, p(y)/q(y))) = \sum_y \max(0, q(y) - p(y))$$

Conditional probability of resampling x given rejection:

$$P(\text{resample} = x|\text{reject}) = \frac{\max(0, p(x) - q(x))}{\sum_y \max(0, p(y) - q(y))}$$

Combining both cases:

$$P(\text{output} = x) = \min(p(x), q(x)) + \frac{\max(0, p(x) - q(x))}{Z} \cdot Z = p(x) \quad \square$$

where $Z = \sum_y \max(0, p(y) - q(y)) = \sum_y q(y) - p(y))^- = \|p - q\|_{\text{TV}}$.

70.2 Expected Speedup Analysis

Acceptance Rate

The per-token acceptance rate:

$$\alpha = \mathbb{E}_{x\sim q}[\min(1, p(x)/q(x))] = 1 - \|p - q\|_{\text{TV}} = 1 - \frac{1}{2}\sum_x |p(x) - q(x)|$$

When draft and target agree ($p = q$): $\alpha = 1$ (all tokens accepted). When they disagree maximally: $\alpha = 0$ (no tokens accepted).

Expected accepted tokens per target call. The number of accepted tokens in a batch of K follows a truncated geometric distribution:

$$\mathbb{E}[\text{accepted tokens}] = \frac{1 - \alpha^{K+1}}{1 - \alpha}$$

For $\alpha \to 1$: $\approx K$ (all accepted). For $\alpha \to 0$: ≈ 1 (one token per call).

Latency model. Let c_{target} be the cost (latency) per target model forward pass and c_{draft} per draft token generation. One speculative decoding iteration:

$$t_{\text{spec}} = K \cdot c_{\text{draft}} + c_{\text{target}}$$

producing on average $\frac{1-\alpha^{K+1}}{1-\alpha}$ tokens. Standard decoding produces 1 token per c_{target} latency. Speedup:

$$S = \frac{c_{\text{target}}}{t_{\text{spec}}/\mathbb{E}[\text{accepted}]} = \frac{c_{\text{target}} \cdot \frac{1-\alpha^{K+1}}{1-\alpha}}{K c_{\text{draft}} + c_{\text{target}}}$$

For $c_{\text{draft}} \ll c_{\text{target}}$ and $\alpha = 0.9$, $K = 5$: $S \approx \frac{1-0.9^6}{1-0.9} \approx \frac{0.469}{0.1} = 4.69\times$ speedup.

Example — Speculative decoding: Llama-3-70B with Llama-3-8B draft

Target: Llama-3-70B, $c_{\text{target}} = 80$ ms/token (A100 fp16). Draft: Llama-3-8B, $c_{\text{draft}} = 5$ ms/token. Typical acceptance rate: $\alpha = 0.82$ (from empirical evaluation on coding tasks). Draft batch: $K = 5$.
$\mathbb{E}[\text{accepted}] = (1-0.82^6)/(1-0.82) = (1-0.305)/0.18 = 3.86$ tokens. $t_{\text{spec}} = 5\times5+80 = 105$ ms for 3.86 tokens. Speed: $3.86/105 = 0.0368$ tokens/ms.
Standard: $1/80 = 0.0125$ tokens/ms. Speedup: $0.0368/0.0125 = 2.94\times$.

70.3 Self-Speculative Decoding

Zhang et al. (2023) propose using the same model as both draft and target but skipping certain layers for the draft:

- Draft: run layers $\{1, 2, \ldots, L/2\}$ only (half the layers).
- Verify: run all L layers, processing the draft outputs in batch.

The acceptance rate is higher than a separate small model (same weights, just fewer layers) but the draft cost reduction is smaller (layer-skipping reduces cost by only $L/2$, not the full small-model savings). For transformers, layer-skipping reduces cost approximately linearly: $c_{\text{draft}} \approx c_{\text{target}}/2$.

This is especially useful when a smaller model of the same family is not available.

PART 71

Mixture of Depths and Adaptive Computation

Not all tokens require the same depth of processing. Mixture of Depths routes tokens dynamically to different numbers of transformer layers. We derive the routing objective, the relationship to early exit mechanisms, and the theoretical compute savings.

71.1 Mixture of Depths (MoD)

Raposo et al. (2024) propose that tokens should be processed by a *variable* number of transformer layers, using a learned router to decide which tokens skip which layers.

MoD Layer

At layer ℓ, for each token t, compute a routing scalar $r_t^\ell \in \mathbb{R}$:

$$r_t^\ell = \mathbf{w}_r^\ell \cdot \mathbf{x}_t^\ell$$

Select the top-C tokens (capacity $C = \lceil \text{cap_frac} \cdot n \rceil$) by routing score for processing. Remaining tokens *skip* layer ℓ (residual identity):

$$\mathbf{x}_t^{\ell+1} = \begin{cases} \mathbf{x}_t^\ell + \text{Layer}^\ell(\mathbf{x}_t^\ell) & t \in \text{Top-}C \\ \mathbf{x}_t^\ell & \text{otherwise} \end{cases}$$

Compute savings. A standard transformer with L layers processes all n tokens at every layer: $O(Ln^2d)$ for attention. MoD with capacity fraction $\text{cap} = 0.5$ processes only $0.5n$ tokens per layer: $O(L \times 0.5n^2 d) = O(Ln^2d/2)$ in attention, plus routing overhead $O(Lnd)$.

Isoflop comparison. With fixed compute budget, MoD allows either:

- Same architecture, $2\times$ longer sequence length, or
- Deeper model (more layers L) at same sequence length, or
- Larger d (wider model) at same length.

Empirically, MoD achieves better loss per FLOP than vanilla transformers because easier tokens (common words, punctuation) require few layers while harder tokens (rare words, long-range dependencies) receive more processing.

71.2 Early Exit

An alternative to MoD: allow the model to exit at an intermediate layer $\ell^* < L$ if a confidence threshold is met.

Early Exit Criterion

At layer ℓ, compute exit score:

$$c^\ell = \text{softmax}(\mathbf{W}_U \mathbf{h}^\ell) \quad \in \Delta^{V-1}$$

Exit if $\max_v c_v^\ell \geq \tau$ (top prediction confidence exceeds threshold τ).

Expected compute. For a distribution over exit layers $P(\ell^* = \ell)$:

$$\mathbb{E}[\text{FLOPs}] = \sum_{\ell=1}^{L} P(\ell^* = \ell) \cdot \ell \cdot C_{\text{layer}}$$

If most tokens exit at layer $\ell^*/L \approx 0.3$: $\approx 30\%$ of standard compute.

Relationship to MoD. MoD skips entire layers for subsets of tokens; early exit exits a *full* position early. MoD is better for batch processing (all tokens participate in every layer boundary, enabling efficient CUDA kernels); early exit can create variable-length batches.

71.3 Adaptive Computation Time (ACT)

Graves (2016) introduced Adaptive Computation Time for RNNs: the model learns how many steps to "think" before producing an output.

For transformers, ACT generalises to the "halting probability" $h_t^\ell \in (0, 1)$ at each layer:

$$h_t^\ell = \sigma(\mathbf{w}_h \cdot \mathbf{x}_t^\ell + b_h)$$

$$N_t = \min\left\{ n : \sum_{\ell=1}^{n} h_t^\ell \geq 1 - \epsilon \right\}$$

The model continues processing token t for N_t steps. The ACT loss penalises excessive computation:

$$\mathcal{L}_{\text{ponder}} = \mathcal{L}_{\text{task}} + \beta \sum_t (N_t + R_t)$$

where $R_t = 1 - \sum_{\ell=1}^{N_t - 1} h_t^\ell$ is the remainder. This encourages the model to use fewer steps while maintaining task performance.

PART 72

Knowledge Distillation: Temperature Scaling, Feature Matching and Contrastive Distillation

We derive the soft label loss for knowledge distillation, prove that KD minimises a KL divergence under specific conditions, and analyse feature-level distillation methods for compressing transformers.

72.1 Hinton's Knowledge Distillation

Knowledge Distillation (KD) Loss

Let $f_T(x)$ and $f_S(x)$ be teacher and student logits. The KD loss:

$$\mathcal{L}_{\text{KD}} = (1-\lambda)\mathcal{L}_{\text{CE}}(y, f_S(x)) + \lambda T^2 \mathcal{L}_{\text{CE}}(\text{softmax}(f_T(x)/T), \text{softmax}(f_S(x)/T))$$

where $T > 1$ is the distillation temperature and y is the hard label.

Why T^2? The soft CE term:

$$T^2 \mathcal{L}_{\text{CE}}(p_T, p_S) = -T^2 \sum_v p_T^{(v)} \log p_S^{(v)}$$

When $T \to \infty$, $p_T^{(v)} \to 1/V$ (uniform). The gradient of the soft CE w.r.t. student logit $f_S^{(v)}$:

$$\nabla_{f_S^{(v)}} \mathcal{L}_{\text{soft}} = T^2(p_S^{(v)} - p_T^{(v)})/T = T(p_S^{(v)} - p_T^{(v)})$$

The T^2 compensates for the $1/T$ factor in the softmax gradient, keeping gradients $O(1)$ regardless of temperature.

KD as KL minimisation. The soft CE loss:

$$T^2 \mathcal{L}_{\text{CE}}(p_T, p_S) = T^2 \left[H(p_T) + D_{\text{KL}}(p_T \| p_S)\right]$$

Since $H(p_T)$ doesn't depend on θ_S, minimising the soft CE is equivalent to minimising $D_{\text{KL}}(p_T \| p_S)$ — forward KL divergence from teacher to student. This is a mass-covering objective: the student distributes mass over all teacher modes.

Information content of soft labels. The teacher's soft label provides more information than the hard label. The information content difference:

$$I_{\text{soft}} - I_{\text{hard}} = \log V - H(p_T) \geq 0$$

For a well-calibrated teacher with entropy $H(p_T) \approx 1$ bit (moderately confident): $I_{\text{soft}} = \log V - 1$ bits vs. $I_{\text{hard}} = 0$ bits. Soft labels provide $(\log V - 1)$ extra bits of signal per training example.

72.2 Feature-Level Distillation

FitNets (Romero et al. 2015) and TinyBERT (Jiao et al. 2019) match intermediate representations:

$$\mathcal{L}_{\text{feature}} = \sum_{\ell \in \mathcal{M}} \|\mathbf{W}_r^{(\ell)} h_S^{(\ell)} - h_T^{(\ell)}\|_F^2$$

where $\mathbf{W}_r^{(\ell)}$ is a learnable projection (needed when student and teacher have different dimensions) and $\mathcal{M}$ is a mapping from student to teacher layers.

Attention distribution distillation. TinyBERT additionally matches attention matrices:

$$\mathcal{L}_{\text{attn}} = \frac{1}{H}\sum_{h=1}^{H} \text{MSE}(A_S^{(h)}, A_T^{(h)})$$

Matching attention patterns is especially effective because attention heads encode interpretable behaviours (induction heads, copy suppression) that are difficult to recover from output distributions alone.

72.3 Structured Pruning

Pruning removes entire components (heads, layers, neurons) rather than individual weights, preserving hardware efficiency.

Head pruning. Each attention head h contributes:

$$\Delta\mathcal{L}_h = \mathcal{L}(W \setminus h) - \mathcal{L}(W)$$

The head importance score $I_h = |\Delta\mathcal{L}_h|$. Approximated via the Taylor expansion:

$$I_h \approx \left| \frac{\partial \mathcal{L}}{\partial \mathbf{x}_h} \cdot \mathbf{x}_h \right|$$

Heads with small importance can be removed. Michel et al. (2019) show that $\approx 70\%$ of attention heads in BERT can be removed with $< 1\%$ accuracy drop on GLUE, because most heads implement redundant or uninformative patterns.

Lottery ticket hypothesis. Frankle & Carlin (2019) show that large networks contain small "winning ticket" subnetworks that, when trained in isolation from the same initialisation, achieve full accuracy. Formally: there exists a mask $m \in \{0,1\}^{|\theta|}$ with $\|m\|_0 \ll |\theta|$ such that:

$$f(x; m \odot \theta_0) \approx f(x; \theta^*_{\text{full}})$$

where θ_0 is the random initialisation and θ^*_{full} is the trained full network.

Connection to double descent. The lottery ticket phenomenon implies that large networks can interpolate training data via many different sparse subnetworks. The overparameterisation enables the lottery ticket to be found by SGD; the implicit regularisation (gradient descent from small initialisation) selects the sparse winner.

Part 73

Continual Learning: Elastic Weight Consolidation and Gradient Projection

We derive Elastic Weight Consolidation (EWC) from a Bayesian perspective, analyse the gradient projection method for preventing interference, and develop the theoretical upper bound on forgetting in the linear case.

73.1 Catastrophic Forgetting

When a neural network is trained sequentially on tasks $\mathcal{T}_1, \mathcal{T}_2, \ldots$, it tends to forget earlier tasks as it adapts to later ones. Formally:

$$\mathcal{L}_{\mathcal{T}_1}(\theta^*_{\mathcal{T}_2}) \gg \mathcal{L}_{\mathcal{T}_1}(\theta^*_{\mathcal{T}_1})$$

where $\theta^*_{\mathcal{T}_k}$ is the optimal parameter after training on $\mathcal{T}_k$.

Why forgetting occurs. The Hessian $H_k = \nabla^2 \mathcal{L}_{\mathcal{T}_k}(\theta^*)$ of task k has a low-dimensional subspace of "important" directions (high curvature) and a large null space of "unimportant" directions (zero curvature). Training on $\mathcal{T}_2$ moves θ in directions that may be irrelevant to $\mathcal{T}_2$ but destroy $\mathcal{T}_1$'s solutions (in the high-curvature directions of H_1).

73.2 Elastic Weight Consolidation (EWC)

Kirkpatrick et al. (2017) derive EWC from Bayesian online learning. After training on $\mathcal{T}_1$, the posterior:

$$p(\theta|\mathcal{D}_1) \propto p(\mathcal{D}_1|\theta)p(\theta) = \exp(-\mathcal{L}_{\mathcal{T}_1}(\theta))p(\theta)$$

Training on $\mathcal{T}_2$ with prior $p(\theta|\mathcal{D}_1)$:

$$\log p(\theta|\mathcal{D}_1, \mathcal{D}_2) \propto -\mathcal{L}_{\mathcal{T}_2}(\theta) + \log p(\theta|\mathcal{D}_1)$$

Approximate $\log p(\theta|\mathcal{D}_1)$ by a second-order Taylor expansion around θ^*_1:

$$\log p(\theta|\mathcal{D}_1) \approx -\frac{1}{2}(\theta - \theta^*_1)^\top F_1 (\theta - \theta^*_1) + \text{const}$$

where $F_1 = -\mathbb{E}_{\mathcal{D}_1}[\nabla^2 \log p(\mathcal{D}_1|\theta^*_1)]$ is the Fisher information matrix at θ^*_1.

$$\mathcal{L}_{\text{EWC}} = \mathcal{L}_{\mathcal{T}_2}(\theta) + \frac{\lambda}{2}\sum_i F_{1,i}(\theta_i - \theta^*_{1,i})^2$$

where $F_{1,i}$ is the diagonal Fisher (tractable approximation to the full FIM).

Interpretation. Parameters with high Fisher information ($F_{1,i}$ large) are important for task 1 and are penalised heavily for moving away from $\theta^*_{1,i}$. Parameters with low Fisher information can move freely.

Fisher as task importance. The diagonal Fisher:

$$F_{1,i} = \mathbb{E}_{\mathcal{D}_1}\left[\left(\frac{\partial \log p(y|x,\theta^*_1)}{\partial \theta_i}\right)^2\right]$$

High $F_{1,i}$ means parameter i strongly influences the output distribution on task 1 (the squared gradient is large on average).

73.3 Gradient Projection

Saha et al. (2021) propose projecting gradients orthogonally to the subspace spanned by previous task gradients:

$$g_\perp^{(2)} = g^{(2)} - \text{proj}_{\text{span}(G_1)} g^{(2)}$$

where $G_1 = [g^{(1,1)}, \dots, g^{(1,m)}]$ is a matrix of gradients from task 1.

Projection. $\text{proj}_{G_1} g = G_1 (G_1^\top G_1)^{-1} G_1^\top g$ (assuming G_1 has full column rank). The orthogonal component:

$$g_\perp = (I - G_1 (G_1^\top G_1)^{-1} G_1^\top) g = (I - P_{G_1}) g$$

where P_{G_1} is the projection onto $\text{col}(G_1)$.

Forgetting bound. For a quadratic task 1 loss $\mathcal{L}_1(\theta) = \frac{1}{2}\|\theta - \theta_1^*\|^2$ and gradient step $\theta \leftarrow \theta - \eta g_\perp^{(2)}$:

$$\mathcal{L}_1(\theta') - \mathcal{L}_1(\theta) = -\eta (\theta - \theta_1^*)^\top g_\perp^{(2)} + O(\eta^2)$$

Since $g_\perp^{(2)} \perp (\theta - \theta_1^*)$ only if $(\theta - \theta_1^*) \in \text{col}(G_1)$, the bound is: $|\mathcal{L}_1(\theta') - \mathcal{L}_1(\theta)| \le \eta \|\theta - \theta_1^*\| \|g_\perp^{(2)}\| \cdot \sin(\angle(g_1, g_2))$.

Gradient projection minimises the angle between the gradient and the constraint hyperplane, keeping $\mathcal{L}_1$ approximately constant during task 2 training.

Part 74

Tokenisation: BPE, Unigram LM and the Information-Theoretic Optimum

We derive the Byte Pair Encoding algorithm as a greedy approximation to MDL, the Unigram Language Model tokeniser as EM on a latent variable model, and establish the connection between vocabulary size and model capacity.

74.1 Tokenisation as Data Compression

A tokeniser maps text $s \in \Sigma^*$ (characters) to a token sequence $t \in \mathcal{V}^*$ (vocabulary elements). Minimising token sequence length maximises token density of information.

Information-theoretic optimum. The optimal tokeniser minimises the expected number of tokens per character:

$$\text{tokens per char} = \frac{\mathbb{E}[|t(s)|]}{\mathbb{E}[|s|]}$$

Equivalently, it maximises the average bits per token. By Shannon's source coding theorem, the minimum achievable code length per character is $H(\mathcal{P})$ where $\mathcal{P}$ is the character distribution. The optimal tokenisation assigns length $-\log_2 p(w)$ to token w (Huffman coding).

74.2 Byte Pair Encoding (BPE)

BPE (Sennrich et al. 2016) builds a vocabulary greedily:

1. Initialise vocabulary with all characters.
2. Count all adjacent pair frequencies in the corpus.
3. Merge the most frequent pair, adding the merged bigram to the vocabulary.
4. Repeat for $V - V_0$ merges (reaching target vocabulary size V).

MDL interpretation. Each merge step reduces the total description length of the corpus by (approximately) the savings from using one symbol for the merged pair:

$$\Delta\text{MDL} = f(ab) \cdot (\text{len}(a) + \text{len}(b) - \text{len}(ab))$$

In the limit of optimal encoding, $\text{len}(x) = -\log_2 P(x)$ bits. Since $\text{len}(ab) < \text{len}(a) + \text{len}(b)$ whenever $P(ab) > P(a)P(b)$ (positive PMI), BPE greedily selects pairs with highest positive PMI.

74.3 Unigram Language Model Tokeniser

SentencePiece Unigram (Kudo 2018) maintains a vocabulary $\mathcal{V}$ and a unigram model $P(w)$ for each token, finding the most probable segmentation.

Segmentation as a latent variable. For text s, let $t(s)$ be a segmentation. The model probability:

$$P(s|\mathcal{V}, P) = \max_{t:\text{concat}(t)=s} \prod_{w \in t} P(w)$$

EM training.

- E-step: compute the probability of each segmentation using the Viterbi algorithm.

- M-step: update $P(w)$ as the normalised expected count: $P(w) \propto \sum_s \mathbb{E}_{P(t|s)}[\text{count}(w,t)]$.

Vocabulary reduction. To reduce vocabulary from $|\mathcal{V}|$ to V_{target}: compute the loss increase ΔL_w if token w is removed (characters fill the gap). Remove the $|\mathcal{V}| - V_{\text{target}}$ tokens with lowest ΔL_w.

74.4 Vocabulary Size and Model Capacity

Embedding layer cost. The embedding matrix $\mathbf{W}_E \in \mathbb{R}^{V \times d}$ has Vd parameters. For $V = 100{,}000$ and $d = 4096$: 4×10^8 parameters (a significant fraction of the model for small d).

Tradeoff. Larger vocabulary: (a) fewer tokens per sentence (longer-range dependencies become shorter), (b) more parameters in embedding, (c) rarer tokens have fewer training examples.

Empirical optimal. Tao et al. (2024) find that for a transformer with N parameters and D training tokens, the optimal vocabulary size:

$$V^* \approx 0.1 N^{0.5} D^{0.1}$$

For $N = 7B$, $D = 10^{12}$: $V^* \approx 0.1 \times 83666 \times 10^{10 \times 0.1} = 0.1 \times 83666 \times 1.585 \approx 13{,}250$ tokens — surprisingly small compared to GPT-4's reported 100K.

PART 75 — FINAL SYNTHESIS

Volume V in Retrospect: The Mathematical Unity of Large-Scale Neural Networks

We draw together the themes of this volume — efficient inference, emergent reasoning, mechanistic structure, and alignment mathematics — into a coherent mathematical picture of large-scale neural networks at the 2024–2025 research frontier.

75.1 The Three Axes of Modern Neural Network Mathematics

The mathematics developed in this volume can be organised along three axes:

Axis 1: Computational Efficiency. The quadratic attention bottleneck drove the entire ecosystem of approximations: sliding window attention ($O(nw)$), GQA/MLA (KV cache compression), SSMs ($O(n)$ recurrence), linear attention (FAVOR+, RWKV), and hardware-aware algorithms (FlashAttention, speculative decoding). The unifying mathematical framework is the *matrix compression* perspective: every efficient attention variant replaces the $n \times n$ attention matrix with a structure that can be computed, stored, and applied in sub-quadratic time.

Axis 2: Emergent Capabilities and Scale. The Chinchilla scaling laws provide a deterministic equation for model quality as a function of compute allocation. Emergence arises at phase transitions in this loss landscape — either from metric artefacts (Schaeffer) or genuine algorithmic thresholds (Telgarsky's depth separation, the modular arithmetic Fourier circuit). The mathematical tools are statistical physics (phase transitions, order parameters) and circuit complexity (TC^0, NC^1 circuits).

Axis 3: Alignment and Control. The RLHF pipeline — Bradley-Terry reward modelling, KL-constrained optimisation, PPO/DPO/GRPO — formalises the alignment problem as reward learning under distribution shift. The fundamental tension is the overoptimisation/reward hacking tradeoff: more RL gives higher proxy reward but lower gold reward past a critical KL divergence, controlled by the Pinsker inequality.

75.2 Unifying Formula: The Efficiency-Quality-Alignment Triad

For a deployed language model serving n_q queries per day:

$$\text{Total cost} = C_{\text{train}} + n_q \left(N \cdot t_{\text{layer}} \cdot L_{\text{eff}} \right) + C_{\text{human_feedback}}$$

where:

- $C_{\text{train}} = 6N^*D^*$ (Chinchilla-optimal training).
- $N \cdot t_{\text{layer}} \cdot L_{\text{eff}}$: inference cost. $L_{\text{eff}} = L \cdot (1 - \text{skip_rate})$ for MoD; $L_{\text{eff}} = L/S$ for speculative decoding speedup S; t_{layer} scales with KV cache size for memory-bound inference.
- $C_{\text{human_feedback}}$: cost of preference data (reduced by β-CAI and GRPO).

Every technique in this volume reduces one term:

- Scaling laws: minimise C_{train} per quality point.
- Efficient attention, GQA, MoD: minimise t_{layer}.
- Speculative decoding: minimise L_{eff} by batching verifications.

- RLHF/GRPO: maximise quality per unit $C_{\text{human_feedback}}$.
- LoRA/QLoRA: reduce C_{train} for fine-tuning.

75.3 Open Mathematical Questions

1. **Optimal inference-time compute allocation.** Given a fixed training compute C_{train} and query budget n_q, what is the joint optimum $(N^*, D^*, T^*_{\text{inf}})$? The Chinchilla answer ignores inference cost; the full solution requires a three-way optimisation that remains open.
2. **Mechanistic completeness.** Do the identified circuits (induction heads, IOI circuit, Fourier modular arithmetic) constitute a complete mechanistic account of language model behaviour, or are there fundamental capabilities that resist circuit-level decomposition? This is the "interpretability completeness" conjecture.
3. **Why does in-context learning work?** The Bayesian ICL hypothesis (Xie et al.) provides a plausible story but lacks a derivation from first principles. Under what conditions does pretraining on a mixture of tasks induce a posterior that correctly identifies the task from a few examples? The answer likely involves the concept of "implicit Bayes-optimal predictors" in the training distribution.
4. **Solvability of the reward hacking problem.** Is there a provably safe RLHF algorithm that achieves the gold reward's maximum while guaranteeing no overoptimisation? The current best bound (from Pinsker) allows $O(\sqrt{\text{KL}})$ reward model error. Tighter bounds would require assumptions on the structure of the reward model's errors.
5. **The emergence-from-pretraining question.** Why does next-token prediction on natural language text create representations that are linearly probed for world-model features (days of week, spatial positions, truth values)? A rigorous answer would connect the information content of natural language to the functional form of transformer representations, likely via some version of the Information Bottleneck principle applied to the pretraining task.

Closing note. The mathematical framework of this series — differential geometry, information theory, statistical learning theory, group representation theory, circuit complexity — provides the tools to formalise these questions precisely. The empirical observations of the 2020–2025 era (scaling laws, emergent capabilities, in-context learning, mechanistic circuits) are now sufficiently documented to admit rigorous mathematical treatment. The next decade of neural network mathematics will likely be characterised by the derivation of these empirical laws from first principles, analogous to how the empirical regularities of thermodynamics were explained by statistical mechanics. The mathematical foundations laid in these five volumes are the starting point.

PART 53 — MOE SUPPLEMENT

MoE Routing: Load Balance Theory and Expert Specialisation

We analyse MoE routing mathematically via the lens of optimal transport, derive the load-balance auxiliary loss from information-theoretic principles, and prove conditions under which expert specialisation is stable at equilibrium.

53.10 MoE as Optimal Transport

Consider the token-to-expert assignment problem for a batch of M tokens and E experts. Each token i has a routing probability vector $\mathbf{g}^{(i)} \in \Delta^{E-1}$ (simplex). Top-k routing assigns each token to k experts with highest $g_e^{(i)}$.

Optimal transport formulation. Define a cost matrix $\mathbf{C} \in \mathbb{R}^{M \times E}$ where $C_{ie} = -g_e^{(i)}$ (negative routing logit, so lower cost = higher preference). The token-expert assignment $\mathbf{T} \in \mathbb{R}_{\geq 0}^{M \times E}$ satisfies:

- Row marginal: $\sum_e T_{ie} = k/E$ (each token uses k experts on average).
- Column marginal: $\sum_i T_{ie} = Mk/E$ (each expert receives equal load).
- Minimum cost: $\min_{\mathbf{T}} \langle \mathbf{C}, \mathbf{T} \rangle_F$.

This is a balanced transportation problem. The standard top-k routing approximates this with a greedy assignment, which is optimal only when $k = 1$ (Hungarian algorithm). For $k > 1$, the globally optimal assignment requires solving the LP.

Sinkhorn routing. Riquelme et al. (2021) propose solving the balanced assignment via Sinkhorn's algorithm (iterative proportional fitting):

1. Initialise $\mathbf{T} = \exp(-\mathbf{C}/\tau)$.
2. Normalise rows: $T_{ie} \leftarrow T_{ie} / \sum_{e'} T_{ie'}$.
3. Normalise columns: $T_{ie} \leftarrow T_{ie} / \sum_{i'} T_{i'e}$.
4. Repeat for R iterations.

Sinkhorn's algorithm converges to the entropy-regularised optimal transport plan. The entropy term $\tau H(\mathbf{T})$ encourages soft assignments, smoothing the discrete top-k decision.

53.11 Expert Specialisation at Equilibrium

Under what conditions do MoE experts specialise to different token types?

Theorem (informal, Zuo et al. 2022). In a two-expert MoE ($E = 2$) with top-1 routing, if the data distribution $p(x)$ is a mixture of two clusters p_1, p_2 with separation $\Delta = \|(\mu_1 - \mu_2)\| > \sqrt{d/n}$ (where d is the input dimension and n is the dataset size), then the trained router assigns cluster 1 to expert 1 and cluster 2 to expert 2 with probability $1 - \exp(-\Omega(n\Delta^2/d))$.

Proof sketch. The router's logit difference $g_1(x) - g_2(x) = \mathbf{w}^\top x$ is a linear function. By the data separation condition, a separating hyperplane exists. Gradient descent finds this hyperplane with high probability, routing each cluster to its specialist expert. The probability bound comes from the VC dimension of linear classifiers in $\mathbb{R}^d$.

Expert utilisation. In practice, specialisation is not perfect. The "expert utilisation" $u_e =$

$\sum_i \mathbf{1}[\text{token } i \text{ assigned to } e]/M$ measures how uniformly tokens are distributed. The auxiliary loss $\mathcal{L}_{\text{aux}} = E \sum_e f_e p_e$ (where f_e is observed fraction and p_e is mean routing probability) penalises the correlation between actual and predicted load, encouraging $f_e \to 1/E$ for all e.

Information-theoretic bound on load imbalance. The maximum possible specialisation (entropy of routing distribution) is bounded by:

$$H(\text{routing}) \leq \log E$$

Perfect load balance achieves $H = \log E$ (uniform assignment). Expert collapse achieves $H = 0$ (all tokens to one expert). The auxiliary loss implicitly maximises $H(\text{routing})$.

Example — Load balance in Mixtral 8×7B

With $E = 8$ experts and top-2 routing, the routing entropy is $H \approx \log 8 = 3.0$ bits at perfect balance.
Empirical measurement (Jiang et al. 2024): $H \approx 2.7$ bits — 90% of maximum entropy, indicating near-uniform utilisation with some specialisation.
Most utilised expert: handles $\approx 16\%$ of tokens (vs. 12.5% at perfect balance). Domain analysis shows this expert specialises in arithmetic and code, consistent with the specialisation theorem.

PART 55 — SPECTRAL SUPPLEMENT

SSMs in the Frequency Domain: Spectral Analysis and Audio Modelling

We derive the frequency response of S4 and Mamba, analyse their expressiveness in modelling long-range dependencies via spectral leakage, and compare with the transformer's flat frequency response.

55.9 Frequency Response of SSMs

A linear SSM with state matrix $\bar{\mathbf{A}}$ and input/output matrices $\bar{\mathbf{B}}, \mathbf{C}$ is equivalent to convolution with the kernel:

$$K = (\mathbf{C}\bar{\mathbf{B}},\ \mathbf{C}\bar{\mathbf{A}}\bar{\mathbf{B}},\ \mathbf{C}\bar{\mathbf{A}}^2\bar{\mathbf{B}},\ \ldots)$$

Discrete-time Fourier transform of the kernel. The DTFT of K is:

$$\hat{K}(\omega) = \sum_{k=0}^{\infty} K_k e^{-i\omega k} = \mathbf{C}(e^{i\omega}\mathbf{I} - \bar{\mathbf{A}})^{-1}\bar{\mathbf{B}}$$

For a diagonal SSM with $\bar{\mathbf{A}} = \operatorname{diag}(\lambda_1, \ldots, \lambda_N)$:

$$\hat{K}(\omega) = \sum_{k=1}^{N} \frac{\mathbf{C}_k \bar{\mathbf{B}}_k}{e^{i\omega} - \lambda_k}$$

This is a sum of N Cauchy kernels in the complex plane. Each pole λ_k at distance $|1 - \lambda_k|$ from the unit circle creates a resonance at angular frequency $\angle\lambda_k$.

Long-range dependency via poles near the unit circle. For $|\lambda_k| \to 1$ (eigenvalue close to the unit circle), the corresponding term $1/(e^{i\omega} - \lambda_k)$ has a sharp peak in magnitude at $\omega = \angle\lambda_k$. This implements a selective bandpass filter: the SSM amplifies inputs near frequency $\angle\lambda_k$ and attenuates others.

HiPPO poles. The HiPPO matrix has eigenvalues on the imaginary axis (continuous time), corresponding to poles at $z = 1$ (after discretisation with small Δ). This means all frequencies are amplified equally at the HiPPO initialisation — a flat response that allows the model to learn any frequency weighting from data.

55.10 Audio and Video Modelling with SSMs

Audio sequences. Speech audio at 16 kHz has $n = 16000 \times$ seconds samples per second. A 10-second clip: $n = 160000$ samples. Standard transformers require $O(n^2) = O(2.56 \times 10^{10})$ attention computations — infeasible.

Mamba (state dim $N = 16$, $\Delta \approx 1/16000$) processes audio with $O(Nn)$ cost. The key advantages:

- The input-dependent Δ_k allows the model to focus on high-energy periods (speech) and fast-forward through silence.
- The selective state update $\mathbf{B}_k$ allows the state to selectively encode phoneme-level features rather than raw samples.

Video sequences. At T frames of resolution $H \times W$, the total token count is $T \times H/P \times W/P$ where P is the patch size. For $T = 300$ frames (10 seconds at 30 fps), $H = W = 224$,

$P = 16$: $n = 300 \times 14 \times 14 = 58800$ tokens. SSM processing cost: $O(Nn)$ with small constant; transformer: $O(n^2) \approx 3.5 \times 10^9$.

SSMs also naturally handle the temporal structure of video without any special position encoding — the recurrent state accumulates temporal context automatically.

PART 57 — ARCHITECTURE PARAMETER SUPPLEMENT

Transformer Parameter Counting and FLOP Analysis

We derive exact FLOP and parameter counts for all transformer components, establish the 6ND approximation for training compute, and analyse the breakdown by component type across model families.

57.10 Exact Parameter Count

For a transformer with L layers, model dimension d, H attention heads, FFN expansion factor e_{FFN} (typically 4), and vocabulary V:

Attention per layer.

- $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V \in \mathbb{R}^{d \times d}$: $3d^2$ parameters each.
- $\mathbf{W}_O \in \mathbb{R}^{d \times d}$: d^2 parameters.
- Total per layer: $4d^2$ (plus biases if used: $+4d$, usually omitted in large models).

FFN per layer. With SwiGLU activation (standard in LLaMA-3, Gemma): $\mathbf{W}_1, \mathbf{W}_3 \in \mathbb{R}^{e_{\text{FFN}} d \times d}$ and $\mathbf{W}_2 \in \mathbb{R}^{d \times e_{\text{FFN}} d}$: $3e_{\text{FFN}} d^2$ parameters per layer.

For standard FFN (GPT-style): $2e_{\text{FFN}} d^2$.

Layer norm. $2d$ parameters per norm (gain and bias), 2 norms per layer: $4d$.

Total (SwiGLU, L layers):

$$N = L(4d^2 + 3e_{\text{FFN}} d^2 + 4d) + Vd + d + \text{positional} \approx L(4 + 3e_{\text{FFN}})d^2 + Vd$$

For $e_{\text{FFN}} = \frac{8}{3}$ (LLaMA convention, rounded to nearest multiple of 256): $N \approx L \cdot 12d^2 + Vd$.

EXAMPLE — LLaMA-3-8B parameter breakdown

$L = 32$, $d = 4096$, $H = 32$, $d_k = 128$, $e_{\text{FFN}} = 14336/4096 \approx 3.5$, $V = 128256$.
Attention per layer: $4 \times 4096^2 = 67.1M$. Total: $32 \times 67.1M = 2.15B$.
FFN per layer (SwiGLU): $3 \times 14336 \times 4096 = 176.2M$. Total: $32 \times 176.2M = 5.64B$.
Embedding: $128256 \times 4096 = 525.3M$.
Layer norms: $32 \times 4 \times 4096 = 524K$.
Total $\approx 2.15 + 5.64 + 0.525 + 0.0005 = 8.32B$ (vs. reported 8.03B — small discrepancy from GQA: $H_k = H_v = 8$ reduces KV projection parameters by $4\times$).

57.11 FLOP Analysis

For a forward pass on sequence length n, batch size 1:

Attention.

- QKV projections: $3 \times 2nd^2 = 6nd^2$ FLOPs (matrix-vector products).
- $\mathbf{QK}^\top$: $2n^2 d$ FLOPs (outer product and dot products).
- Softmax and $\mathbf{AV}$: $2n^2 d$ FLOPs.
- Output projection: $2nd^2$ FLOPs.

- Total: $(8nd^2 + 4n^2d)$ per layer.

FFN (SwiGLU).

- Three matmuls: $2 \times 3 \times n \times d \times e_{\text{FFN}}d = 6ne_{\text{FFN}}d^2$.
- SiLU and elementwise product: $O(ne_{\text{FFN}}d)$ (negligible).
- Total: $\approx 6ne_{\text{FFN}}d^2$.

Total forward FLOPs per layer.

$$C_{\text{layer}} = 8nd^2 + 4n^2d + 6ne_{\text{FFN}}d^2 = (8 + 6e_{\text{FFN}})nd^2 + 4n^2d$$

For $n \ll d$ (batch inference, short sequences): $C_{\text{layer}} \approx (8 + 6e_{\text{FFN}})nd^2$. For $n \gg d$ (long context): $C_{\text{layer}} \approx 4n^2d$ (attention dominates).

The 6ND rule. For a complete training forward + backward pass on D tokens:

$$C_{\text{train}} \approx 6ND$$

where N is the model parameter count. The factor $6 = 2(\text{forward}) + 4(\text{backward})$ (backward costs $\approx 2\times$ forward due to storing activations).

Derivation. Each parameter participates in approximately 2 multiply-adds per forward pass token (once in the weight matrix multiplication that creates it, once in the gradient computation). With N parameters and D tokens: $2ND$ FLOPs forward + $4ND$ backward $= 6ND$.

PART 76

Calibration, Uncertainty and Out-of-Distribution Detection

Well-calibrated models assign correct confidence levels to their predictions. We derive the Expected Calibration Error (ECE), analyse temperature scaling as post-hoc calibration, and formalise out-of-distribution detection via the maximum softmax probability baseline.

76.1 Calibration

Calibration (Perfect)

A model is *perfectly calibrated* if for all confidence levels $c \in [0, 1]$:

$$P(\hat{y} = y \mid \max_k \hat{p}_k = c) = c$$

Among all inputs where the model assigns confidence c to its top prediction, exactly a fraction c of those predictions are correct.

Expected Calibration Error (ECE). Partition predictions into M confidence bins $B_m = (m/M, (m+1)/M]$. Let $\text{acc}(B_m)$ and $\text{conf}(B_m)$ be the mean accuracy and mean confidence within bin m:

$$\text{ECE} = \sum_{m=1}^{M} \frac{|B_m|}{n} |\text{acc}(B_m) - \text{conf}(B_m)|$$

For a perfectly calibrated model, $\text{ECE} = 0$. Modern neural networks are typically overconfident: $\text{conf}(B_m) > \text{acc}(B_m)$ for high-confidence bins. ResNet-110 on CIFAR-100: ECE $\approx 16.9\%$ before calibration.

76.2 Temperature Scaling

Temperature scaling (Guo et al. 2017) rescales logits by a single parameter $T > 0$:

$$\hat{p}_k = \frac{\exp(f_k / T)}{\sum_{k'} \exp(f_{k'} / T)}$$

For $T > 1$: softmax distribution is flatter (less confident). For $T < 1$: sharper (more confident).

Post-hoc calibration. T is learned on a held-out calibration set by minimising the negative log-likelihood:

$$T^* = \arg\min_T \sum_i -\log \hat{p}_{y_i}(T) = \arg\min_T \text{NLL}(T)$$

Since $\text{NLL}(T)$ is a convex function of T (log-sum-exp is convex), the optimum can be found by binary search or gradient descent.

Why temperature scaling works. The maximum likelihood estimator for logistic regression is consistent but not calibrated in the finite sample regime. Large neural networks tend to overfit the softmax probabilities to training data, assigning near-zero probability to non-predicted classes. Temperature scaling "inflates" the uncertainty without changing the argmax (the predicted class).

Example — Temperature scaling on ImageNet

ResNet-50, validation ECE before: 4.5%. After temperature scaling: 1.2%. Optimal $T^* = 1.3$ (slightly higher temperature, slightly less confident).
The calibration improvement is substantial despite using only a single parameter — evidence that miscalibration is primarily a global overconfidence, not a class-specific effect.

76.3 Out-of-Distribution Detection

OOD Detection Problem

Given in-distribution data $\mathcal{D}_{\text{in}}$ and out-of-distribution data $\mathcal{D}_{\text{out}}$ (different domain or corrupted inputs), construct a score function $s(x)$ that is high for $x \in \mathcal{D}_{\text{in}}$ and low for $x \in \mathcal{D}_{\text{out}}$.

Maximum Softmax Probability (MSP) baseline. Hendrycks & Gimpel (2017):

$$s_{\text{MSP}}(x) = \max_k \hat{p}_k(x)$$

In-distribution inputs tend to have high maximum confidence; OOD inputs spread confidence across classes (lower maximum).

Limitation. Neural networks can assign arbitrarily high confidence to OOD inputs, especially far from the training manifold. Lee et al. (2018) show that for x far from all training data, softmax confidence $\to 1$ as $\|x\| \to \infty$ (for logistic regression with ReLU activations) — the exact opposite of desired behaviour.

Mahalanobis distance. A better score using the class-conditional Gaussian fit to each class's penultimate-layer activations $h(x)$:

$$s_{\text{Maha}}(x) = \max_k (h(x) - \hat{\mu}_k)^\top \hat{\Sigma}^{-1} (h(x) - \hat{\mu}_k)$$

where $\hat{\mu}_k, \hat{\Sigma}$ are the class mean and shared covariance. This measures the distance from x to the nearest class in the learned feature space.

Energy score. Liu et al. (2020) propose:

$$E(x; T) = -T \log \sum_k \exp(f_k(x)/T)$$

the (negative) log-partition function of the softmax. This is the free energy in statistical mechanics. For OOD inputs: logits are all near zero, E is small (less negative). For in-distribution inputs: one logit is large, E is large (more negative).

Connection to score matching. The energy score $E(x)$ is exactly the log-normaliser $\log Z$ of the EBM. An EBM trained to assign high probability to in-distribution data assigns $\exp(-E(x))$ as the unnormalised density. OOD detection becomes density estimation: x is in-distribution iff $E(x)$ exceeds a threshold.

Part 77

Message Passing, Graph Transformers and Expressive Power

We formalise message passing neural networks (MPNNs), prove the Weisfeiler-Leman expressiveness limit, and derive Graph Transformers as an attention mechanism on the graph structure.

77.1 Message Passing Neural Networks

MPNN (Gilmer et al. 2017)

For a graph $G = (V, E)$, the MPNN updates node representations $\mathbf{h}_v^{(t)}$:

$$\mathbf{m}_v^{(t+1)} = \mathrm{AGG}\Big(\big\{\mathrm{MSG}^{(t)}(\mathbf{h}_v^{(t)}, \mathbf{h}_u^{(t)}, \mathbf{e}_{vu}) : u \in \mathcal{N}(v)\big\}\Big)$$

$$\mathbf{h}_v^{(t+1)} = \mathrm{UPD}^{(t)}(\mathbf{h}_v^{(t)}, \mathbf{m}_v^{(t+1)})$$

where $\mathcal{N}(v)$ is the neighbourhood, $\mathbf{e}_{vu}$ are edge features, MSG is the message function, AGG is the aggregation (typically sum or max), and UPD is the update function.

Examples.

- GCN: $\mathbf{h}_v^{(t+1)} = \sigma\Big(\mathbf{W}^{(t)} \sum_{u \in \mathcal{N}(v)\cup\{v\}} \frac{\mathbf{h}_u^{(t)}}{\sqrt{d_u d_v}}\Big)$ where $d_v = |\mathcal{N}(v)| + 1$.
- GraphSAGE: $\mathbf{h}_v^{(t+1)} = \sigma\Big(\mathbf{W}^{(t)}[\mathbf{h}_v^{(t)}; \mathrm{MEAN}(\{\mathbf{h}_u : u \in \mathcal{N}(v)\})]\Big)$.
- GAT: $\mathrm{MSG} = \alpha_{vu}\mathbf{W}\mathbf{h}_u$, with attention $\alpha_{vu} = \mathrm{softmax}(\mathbf{a}^\top[\mathbf{W}\mathbf{h}_v; \mathbf{W}\mathbf{h}_u])$.

77.2 The Weisfeiler-Leman Bound

Weisfeiler-Leman (WL) Graph Isomorphism Test

The 1-WL test iteratively refines node colours:

$$c_v^{(t+1)} = \mathrm{hash}\big(c_v^{(t)},\ \{\{c_u^{(t)} : u \in \mathcal{N}(v)\}\}\big)$$

where $\{\{\cdot\}\}$ denotes a multiset. Two graphs are deemed non-isomorphic if their colour distributions differ.

Theorem (Xu et al. 2019). Any MPNN with injective aggregation (e.g. sum) and injective update is at most as powerful as the 1-WL test in distinguishing graph structures. Moreover, the GIN (Graph Isomorphism Network) achieves exactly 1-WL power:

$$\mathbf{h}_v^{(t+1)} = \mathrm{MLP}^{(t)}\left((1 + \epsilon^{(t)})\mathbf{h}_v^{(t)} + \sum_{u \in \mathcal{N}(v)} \mathbf{h}_u^{(t)}\right)$$

Proof sketch. Injectivity of AGG means nodes with different neighbour multisets get different representations — equivalent to the WL colour refinement. The sum aggregation (multiplied by the MLP) preserves multiset information; mean or max aggregation loses it (e.g. cannot distinguish $\{1, 2, 3\}$ from $\{1, 1, 2, 3\}$ with mean).

77.3 Graph Transformers

Graph Transformers (Kreuzer et al. 2021; Rampasek et al. 2022) apply attention over all node pairs (not just neighbours), using graph structure as a positional encoding.

Graph Transformer Attention

$$\text{Attn}_{uv} = \frac{(W_Q \mathbf{h}_u)^\top (W_K \mathbf{h}_v) + b_{uv}}{\sqrt{d_k}}$$

where b_{uv} is a structural bias encoding graph structure:

- Shortest path distance: $b_{uv} = \phi(d(u, v))$ for learnable ϕ.
- Adjacency: $b_{uv} = \mathbf{1}[(u, v) \in E] \cdot w$.
- Laplacian PE: $b_{uv} = \lambda_{uv}^\top \mathbf{w}$ for learned Laplacian features.

Expressive power. Graph transformers with global attention exceed 1-WL expressive power because two nodes can be distinguished by all-pairs path structure (which 1-WL cannot capture). GPS++ (Ma et al. 2023) achieves 3-WL power by combining local MPNN layers with global transformer attention.

Scalability. Full graph attention costs $O(n^2)$ for n nodes — the same bottleneck as sequence transformers. For molecular graphs ($n \leq 100$): tractable. For protein-scale graphs ($n \sim 10000$): requires sparse or hierarchical attention.

PART 78

Mixed Precision Training: FP16, BF16 and FP8

We derive the numerical precision requirements for neural network training, analyse loss scaling as a solution to underflow, and develop the mathematical justification for BF16 over FP16 for large model training.

78.1 Floating-Point Arithmetic for Deep Learning

IEEE 754 floating-point formats differ in their exponent and mantissa bits:

Format	Bits	Exponent	Mantissa	Max value
FP32	32	8	23	3.4×10^{38}
FP16	16	5	10	65504
BF16	16	8	7	3.4×10^{38}
FP8 (E4M3)	8	4	3	448
FP8 (E5M2)	8	5	2	57344

The overflow problem with FP16. Gradient magnitudes during training can exceed 65504 (FP16 max) — especially for large learning rates or before learning rate warmup. Overflow produces `NaN` (Not-a-Number), corrupting training.

Loss scaling. Multiply the loss by a scale factor S before backward pass:

$$g_{\text{scaled}} = S \cdot g_{\text{true}} \quad \text{where} \quad g = \nabla_\theta \mathcal{L}$$

Scale S is chosen so $S \cdot \|g\|_\infty \leq 65504$ (no FP16 overflow). After the backward pass, divide by S before the weight update:

$$\theta \leftarrow \theta - \eta(g_{\text{scaled}}/S)$$

Dynamic loss scaling: automatically increase S when no overflow, decrease by $2\times$ when overflow detected.

78.2 BF16 Advantages

BF16 (Brain Float 16) uses the same 8-bit exponent as FP32, giving the same dynamic range (10^{38}) but less precision (7 mantissa bits vs. 23 in FP32).

Why the exponent matters more than mantissa for training. Neural network gradients span many orders of magnitude — from 10^{-6} (late training, small learning rate) to 10^{+2} (early training, large gradients). The exponent determines which order of magnitude is representable; the mantissa determines precision within that order.

Gradient updates require only ~ 3 bits of effective precision: they are rounded to the weight's resolution anyway. But they must not overflow: hence FP16's limited exponent range causes NaN, while BF16's full FP32 exponent range eliminates this issue.

Precision analysis. The relative rounding error for BF16 is: $\epsilon_{\text{BF16}} = 2^{-7} \approx 0.008$ (0.8%). For FP16: $\epsilon_{\text{FP16}} = 2^{-10} \approx 0.001$ (0.1%). Despite 8× more precise mantissa, FP16 is inferior for training because of overflow.

78.3 FP8 Training

FP8 training (Micikevicius et al. 2022) uses 8-bit floats for forward pass weights and activations, enabling 2× throughput over BF16 on H100 Transformer Engine.

The scaling tensor challenge. FP8 with 4 exponent bits (E4M3) has max value 448. Weight tensors can have max values up to $O(d^{1/2}) \approx 64$ (for random init with $d = 4096$); gradient tensors can reach $O(1)$ — both within FP8 range.

But attention matrices $q^\top k/\sqrt{d_k}$ and softmax outputs require careful per-tensor scaling:

$$\hat{A}_{ij} = \text{quant}_8\left(\frac{A_{ij}}{\text{scale}_A}\right) \cdot \text{scale}_A$$

where $\text{scale}_A = \max_{ij} |A_{ij}|/448$. This per-tensor scaling adds overhead but maintains numerical accuracy.

Empirical results. FP8 training achieves $< 0.5\%$ loss degradation vs. BF16 on LLM pretraining benchmarks (LLAMA-scale models), with $1.8\times$ throughput improvement on H100 GPUs.

PART 79

Theoretical Analysis: Expressivity, Identifiability and Training Dynamics

We survey the theoretical understanding of transformers: their expressivity in circuit complexity terms, identifiability of attention patterns, and the convergence of gradient descent under the mean-field limit.

79.1 Transformer Expressivity

Constant-depth TC^0 circuits. Merrill & Sabharwal (2023) prove that transformers with bounded precision (using p-bit floats) are TC^0 circuits. Specifically:

- Each attention layer corresponds to a threshold circuit of depth $O(1)$.
- A L-layer transformer with $O(L)$ total depth computes a function in TC^0.
- Problems not in TC^0 (believed: integer multiplication, counting) cannot be solved by bounded-precision transformers without chain-of-thought.

The precision caveat. With unbounded precision, transformers can simulate Turing machines. In practice, GPU arithmetic uses FP32 or BF16 — finite precision — so the bounded model is relevant. However, n-digit precision (with n growing with problem size) escapes the TC^0 bound, suggesting that chain-of-thought with sufficient scratchpad length can break the limitation.

Strong expressivity. Despite the TC^0 bound, transformers in practice express a remarkably large class of functions. The resolution: many apparently hard problems (multi-digit addition, pattern recognition) have efficient "shortcut" implementations once sufficient context is available.

79.2 Identifiability of Attention Patterns

Question. Given two transformers T and T' with different weight matrices $\mathbf{W}$ and $\mathbf{W}'$, can we determine whether they compute the same function from their input-output behaviour?

Non-identifiability. Attention is invariant to certain weight transformations. For a head with QKV weights $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V, \mathbf{W}_O$:

- Rotation invariance: $(\mathbf{R}\mathbf{W}_Q, \mathbf{R}\mathbf{W}_K)$ gives the same attention pattern for any orthogonal $\mathbf{R}$.
- Scale invariance: $(\alpha\mathbf{W}_Q, \alpha^{-1}\mathbf{W}_K)$ gives the same attention (scales cancel in the dot product).

The space of equivalent weight configurations is a continuous group — the attention pattern is not uniquely identified by the weights.

Practical consequence. When comparing two trained transformers (e.g. to study how fine-tuning changes attention heads), we must account for these symmetries. Canonical forms (QR decomposition of $\mathbf{W}_Q\mathbf{W}_K^\top$) allow meaningful comparison.

79.3 Mean-Field Analysis of Attention Training

Feature learning regime. Unlike in the NTK regime (where features don't move), realistic transformers learn representations during training. Mean-field theory provides a framework for analysing this.

For a single-layer self-attention head with residual connection, the weight update $\Delta \mathbf{W}_V \propto -\nabla_{\mathbf{W}_V} \mathcal{L}$:

$$\Delta \mathbf{W}_V \propto \mathbf{A}^\top \mathbf{X}^\top (\hat{Y} - Y) \mathbf{W}_O^\top$$

where $\mathbf{A}$ is the attention matrix and $\hat{Y} - Y$ is the prediction error. This is a Hebbian-like update: weight $\mathbf{W}_V$ is strengthened for key-value pairs (k_j, v_j) that contributed to correctly predicted positions.

Convergence. Under the mean-field limit ($n \to \infty$ with fixed ratios), the gradient flow dynamics converge to the solution of an integral equation on the continuous attention distribution. Specifically, the attention kernel $A(q, k)$ converges to the optimal kernel for the given task — formally a Nadaraya-Watson kernel regression problem, which has a unique solution under mild conditions.

PART 80

Multimodal Foundation Models: LLaVA, Flamingo Architecture Mathematics

We derive the vision-language alignment objective for LLaVA, analyse the cross-modal attention mechanism, and formalise the connection between image patch tokens and language token representations.

80.1 Vision-Language Alignment

A vision-language model (VLM) processes interleaved sequences of image patches and text tokens. The challenge: image features live in a visual embedding space $\mathbb{R}^{d_v}$ and text features in a language embedding space $\mathbb{R}^{d_l}$, which are not a priori aligned.

LLaVA alignment. LLaVA (Liu et al. 2023) uses a simple linear projection:

$$\mathbf{z}_{\text{img}} = \mathbf{W}_{\text{proj}} \cdot \mathbf{f}_{\text{CLIP}}(I) \in \mathbb{R}^{d_l}$$

where $\mathbf{f}_{\text{CLIP}}(I)$ are CLIP image features and $\mathbf{W}_{\text{proj}} \in \mathbb{R}^{d_l \times d_v}$.

Training stages.

1. *Feature alignment:* Freeze CLIP and LLM. Train only $\mathbf{W}_{\text{proj}}$ on image-text pairs (CC595K). $\mathcal{L} = -\sum_t \log p_\theta(t_i | t_{<i}, \mathbf{z}_{\text{img}})$.
2. *Instruction tuning:* Unfreeze LLM (keep CLIP frozen). Fine-tune on visual instruction following data (LLaVA-Instruct-150K).

Why linear projection works. CLIP trains both image and text encoders with InfoNCE to maximise $I(f_{\text{img}}(I); f_{\text{txt}}(T))$. At the InfoNCE optimum, image and text representations are linearly aligned in expectation: $\mathbb{E}[\mathbf{f}_{\text{img}}(I)|T] \approx A\mathbf{f}_{\text{txt}}(T)$ for some linear A. The projection $\mathbf{W}_{\text{proj}}$ learns this A, achieving alignment without further contrastive training.

80.2 Patch Token Sequences and 2D Positional Structure

Images are divided into $N_p = (H/P)(W/P)$ patches. Each patch becomes a token. The 2D spatial structure is encoded via 2D positional embeddings.

2D RoPE. Extend 1D RoPE to 2D by applying separate rotations for the x and y spatial coordinates:

$$\text{RoPE}^{2D}(\mathbf{q}, (r, c)) = \text{RoPE}(\text{RoPE}(\mathbf{q}[1 : d/2], r), c)\big|_{\text{concatenated}}$$

where the first half of $\mathbf{q}$ encodes row position r and the second half encodes column position c. The inner product preserves relative 2D positions:

$$(\mathbf{R}_{(r_1,c_1)}\mathbf{q})^\top (\mathbf{R}_{(r_2,c_2)}\mathbf{k}) = \mathbf{q}^\top \mathbf{R}_{(r_1-r_2, c_1-c_2)}\mathbf{k}$$

This is used in ViT-based models (e.g. Idefics, Qwen-VL) to preserve 2D spatial awareness.

Receptive field in ViT. ViT with standard (global) attention has receptive field $= n_p$ (all patches attend to all patches) at every layer. ViT with sliding window attention has receptive field $= w$ patches at layer 1, growing by w per layer. For classification tasks, the [CLS] token's receptive field grows to n_p after $\lceil n_p/w \rceil$ layers.

80.3 Cross-Modal Consistency

Vision-language grounding. A VLM correctly "grounds" a phrase if it correctly localises the referred region. Formally: for text phrase t_i with spatial referent region $R_i \subset [H] \times [W]$, the model's attention from t_i to visual patches should concentrate on patches in R_i.

Cross-attention entropy as a grounding score.

$$\text{grounding}(t_i) = -\sum_j A_{t_i,v_j} \log A_{t_i,v_j}$$

Low entropy: the text token attends to a small spatial region (good grounding). High entropy: uniform attention over all patches (poor grounding).

Empirical finding. Vision tokens in the middle layers (4–8 for 12-layer models) have lower cross-attention entropy than early or late layers, suggesting that grounding develops in the middle of the network. This motivates "grounding-aware" fine-tuning that explicitly supervises middle-layer cross-attention patterns.

Part 81

Decoding Strategies: Beam Search, Nucleus Sampling and Repetition Penalties

We derive the mathematical foundations of text decoding algorithms, analyse the quality-diversity tradeoff via entropy regularisation, and formalise repetition penalties as modified energy functions.

81.1 Greedy Decoding and Beam Search

Greedy decoding selects $\hat{x}_t = \arg\max_v p(x_t = v | x_{1:t-1})$ at each step. This maximises the local probability but can miss globally optimal sequences.

Beam Search

Maintain B partial hypotheses $\{h^{(b)}\}_{b=1}^{B}$ (the beam). At each step:

$$\text{score}(h^{(b)}, v) = \log p_\theta(h^{(b)}) + \log p(v | h^{(b)})$$

Expand all $B \cdot |\mathcal{V}|$ extensions, keep top B by score.

Length normalisation. Beam search favours shorter sequences because log-probability is a sum (shorter sequences have fewer negative terms). Normalise by length:

$$\text{score}_{\text{norm}}(h) = \frac{\log p(h)}{|h|^\alpha}$$

for length penalty $\alpha \in [0.5, 0.8]$ (empirically optimal for translation).

Coverage penalty. For machine translation, add a penalty for source tokens not attended to:

$$\text{score}_{\text{cov}}(h) = \text{score}_{\text{norm}}(h) + \beta \sum_j \log \min\left(1, \sum_t \alpha_{t,j}\right)$$

where $\alpha_{t,j}$ is the attention from output position t to source position j. This encourages the model to cover all source tokens.

81.2 Nucleus Sampling

Nucleus (Top-p) Sampling (Holtzman et al. 2020)

For a probability distribution p over vocabulary $\mathcal{V}$, let $\mathcal{V}^{(p)} \subset \mathcal{V}$ be the smallest set satisfying:

$$\sum_{v \in \mathcal{V}^{(p)}} p(v) \geq p_{\text{nucleus}}$$

Sample from the renormalised distribution:

$$p^*(v) = \begin{cases} p(v) / \sum_{u \in \mathcal{V}^{(p)}} p(u) & v \in \mathcal{V}^{(p)} \\ 0 & \text{otherwise} \end{cases}$$

Entropy connection. The size of the nucleus $|\mathcal{V}^{(p)}|$ is a proxy for the distribution's entropy. When the model is confident (one token dominates), the nucleus is small ($|\mathcal{V}^{(p)}| = 1$ at

$p_{\text{nucleus}} = 0.9$). When the model is uncertain (many tokens have significant probability), the nucleus is large ($|\mathcal{V}^{(p)}| \approx 100$).

Nucleus sampling thus adapts the effective vocabulary to the model's confidence: low-entropy steps get greedy decoding, high-entropy steps get diverse sampling.

Top-k vs. nucleus. Top-k sampling (Fan et al. 2018) always samples from the top k tokens. But the right k varies by context: $k = 1$ (greedy) for highly predictable next tokens ("The_" $\to$ "cat"), $k = 100$ for open-ended creative choices. Nucleus adapts automatically.

81.3 Repetition Penalties

Neural language models tend to repeat phrases, especially at higher temperatures. The repetition penalty (Keskar et al. 2019) modifies logits:

$$\hat{p}(v|x_{1:t}) \propto p(v|x_{1:t})^{1/r(v,x_{1:t})}$$

where $r(v, x_{1:t}) = \theta > 1$ if v appears in $x_{1:t}$, else 1.

Exponential decay variant.

$$\hat{p}(v|x_{1:t}) \propto p(v|x_{1:t}) \cdot \prod_{\tau : x_\tau = v} \exp(-\lambda(t - \tau))$$

Tokens that appeared more recently ($t - \tau$ small) are penalised more; the penalty decays exponentially with time, allowing repetition after sufficient context has passed.

Contrastive search (Su et al. 2022). Alternatively, decode to maximise:

$$\hat{x}_t = \arg\max_{v \in \mathcal{V}^{(k)}} \left[(1 - \alpha) \log p(v|x_{1:t-1}) - \alpha \max_{j<t} \text{sim}(h_v, h_{x_j}) \right]$$

where h_v is the hidden representation of candidate v and sim is cosine similarity. The second term penalises tokens whose representations are similar to previously generated tokens, promoting diversity.

PART 82

Numerical Examples: End-to-End Transformer Computation

We trace a complete forward pass through a minimal transformer, computing exact numerical values for every intermediate representation, and verify that the attention mechanism correctly implements the expected behaviour.

82.1 Minimal Transformer (d=4, 1 head, 2 layers)

Consider a transformer with vocabulary $\mathcal{V} = \{A, B, C\}$, dimension $d = 4$, single attention head ($d_k = d_v = 4$), FFN hidden dimension $= 8$, and sequence length $n = 3$ (input: "A B C").

Setup. Token embeddings (random, for illustration):

$$\mathbf{E} = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \quad (A, B, C \text{ rows})$$

Input sequence "A B C" at positions $1, 2, 3$. Initial embeddings + positional:

$$\mathbf{x}_1^{(0)} = (1, 0, 0, 1)^\top, \quad \mathbf{x}_2^{(0)} = (0, 1, 0, -1)^\top, \quad \mathbf{x}_3^{(0)} = (0, 0, 1, 0)^\top$$

(Adding RoPE with $d = 4$: two frequency pairs with $\theta_0 = 1$, $\theta_1 = 0.01$.)

Query, key projections. Let $\mathbf{W}_Q = \mathbf{W}_K = \mathbf{I}_4$ (identity, for clarity). Queries and keys equal the embeddings.

Attention scores (causal mask).

$$\mathbf{A}^{\text{logit}} = \frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{4}} = \frac{1}{2}\mathbf{X}\mathbf{X}^\top$$

$$= \frac{1}{2}\begin{pmatrix} 2 & -1 & 0 \\ -1 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

After causal mask (upper triangle $= -\infty$):

$$\mathbf{A}^{\text{logit}}_{\text{causal}} = \begin{pmatrix} 1 & -\infty & -\infty \\ -0.5 & 1 & -\infty \\ 0 & 0 & 0.5 \end{pmatrix}$$

Softmax (row-wise):

$$A_{1,:} = \text{softmax}([1, -\infty, -\infty]) = (1, 0, 0)$$

$$A_{2,:} = \text{softmax}([-0.5, 1, -\infty]) = \left(\frac{e^{-0.5}}{e^{-0.5} + e^1}, \frac{e^1}{e^{-0.5} + e^1}, 0\right) = (0.182, 0.818, 0)$$

$$A_{3,:} = \text{softmax}([0, 0, 0.5]) = \left(\frac{1}{1 + 1 + e^{0.5}}, \frac{1}{1 + 1 + e^{0.5}}, \frac{e^{0.5}}{1 + 1 + e^{0.5}}\right) = (0.222, 0.222, 0.556)$$

Value computation. Let $\mathbf{W}_V = \mathbf{I}_4$. Values equal embeddings. Attention output:

$$\mathbf{o}_1 = 1.0 \cdot (1, 0, 0, 1)^\top = (1, 0, 0, 1)^\top$$

$$\mathbf{o}_2 = 0.182(1,0,0,1)^\top + 0.818(0,1,0,-1)^\top = (0.182, 0.818, 0, 0.182 - 0.818)^\top = (0.182, 0.818, 0, -0.636)^\top$$

$$\mathbf{o}_3 = 0.222(1,0,0,1)^\top + 0.222(0,1,0,-1)^\top + 0.556(0,0,1,0)^\top = (0.222, 0.222, 0.556, 0)^\top$$

Residual connection. $\mathbf{h}_i^{(1)} = \mathbf{x}_i^{(0)} + \mathbf{o}_i$ (after layer norm, omitted for brevity).

Observation. Position 3 (token C) now has a representation that "knows" it follows A and B (the attention output mixes information from all three positions). Position 2 (token B) has mostly attended to itself but 18% to A.

This numerical example illustrates how attention builds contextualised representations: each token aggregates information from all previous tokens, weighted by learned query-key similarity.

82.2 Attention Pattern Interpretation

The attention matrix A_{ij} can be decomposed into two effects:

Content-based attention. $\mathbf{q}_i^\top \mathbf{k}_j$ large when tokens i and j have similar semantics (high cosine similarity after projection).

Position-based attention. With RoPE, the dot product $(\mathbf{R}_m \mathbf{q})^\top (\mathbf{R}_n \mathbf{k})$ depends on relative position $m - n$. Trained models develop specific patterns:

- Heads that attend to the previous token: $m - n = 1$ dominant.
- Heads that attend to the first token (sink): $n = 1$ dominant regardless of content.
- Induction heads: attend to the token after the previous occurrence of the current token.

Rank factorisation. The attention pattern for head h can be decomposed:

$$A_{ij}^{(h)} \propto \exp\left(\mathbf{x}_i^\top \mathbf{W}_Q^{(h)\top} \mathbf{W}_K^{(h)} \mathbf{x}_j / \sqrt{d_k}\right)$$

The matrix $\mathbf{W}_Q^{(h)\top} \mathbf{W}_K^{(h)} \in \mathbb{R}^{d \times d}$ has rank at most d_k. This low-rank structure means the head can only detect d_k independent pairwise features between any two positions. With $d_k = 64$ and $d = 4096$: the head uses only $64/4096 = 1.6\%$ of the full feature space for its routing decision.

Appendix C: Mathematical Prerequisites for Volume V

Linear algebra.

- Singular Value Decomposition: $\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^\top$, used in LoRA, NTK, spectral norm.
- Eigendecomposition: $\mathbf{A} = \mathbf{Q}\Lambda\mathbf{Q}^{-1}$, used in SSM analysis, Laplacian PE.
- Woodbury identity: $(\mathbf{A}+\mathbf{UCV})^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{U}(\mathbf{C}^{-1}+\mathbf{VA}^{-1}\mathbf{U})^{-1}\mathbf{VA}^{-1}$, used in DPLR S4.
- Equiangular tight frames: maximal packings of unit vectors in $\mathbb{R}^d$.

Probability and information theory.

- Total variation distance: $\mathrm{TV}(P,Q) = \frac{1}{2}\|p-q\|_1$.
- Pinsker's inequality: $\mathrm{TV}(P,Q)^2 \le \frac{1}{2}D_{\mathrm{KL}}(P\|Q)$.
- Fisher information matrix: $F_{ij} = \mathbb{E}[(\partial_i \log p)(\partial_j \log p)]$.
- Mutual information: $I(X;Y) = D_{\mathrm{KL}}(p_{XY}\|p_X p_Y)$.

Optimisation.

- Bilevel optimisation: $\min_\alpha \mathcal{L}_{\text{val}}(w^*(\alpha),\alpha)$ s.t. $w^*(\alpha) = \arg\min_w \mathcal{L}_{\text{train}}(w,\alpha)$.
- Sinkhorn's algorithm: iterative row/column normalisation for optimal transport.
- Frank-Wolfe method: constrained optimisation over convex sets via linear minimisation oracles.

Complexity theory.

- TC^0: constant-depth polynomial-size threshold circuits.
- NC^1: log-depth polynomial-size Boolean circuits.
- P: polynomial-time Turing machine decisions.
- WL hierarchy: k-WL graph isomorphism tests of increasing power.

Differential geometry.

- Riemannian metric: $g_{ij}(x)dx^i dx^j$ on a manifold.
- Natural gradient: $\tilde{\nabla}_\theta \mathcal{L} = F(\theta)^{-1}\nabla_\theta \mathcal{L}$.
- Lie group: smooth group with compatible manifold structure (e.g. $SO(3)$, $SE(3)$).

Statistical learning theory.

- Rademacher complexity: $\mathcal{R}_n(\mathcal{F}) = \mathbb{E}_\sigma[\sup_{f\in\mathcal{F}} \frac{1}{n}\sum_i \sigma_i f(x_i)]$.
- PAC-Bayes bound: $\mathcal{L}(Q) \le \hat{\mathcal{L}}(Q) + \sqrt{\frac{D_{\mathrm{KL}}(Q\|P)+\ln(2/\delta)}{2n}}$.
- VC dimension: maximum set size that a hypothesis class can shatter.
- Bias-variance decomposition: $\mathbb{E}[(\hat{f}-f^*)^2] = \text{Bias}^2 + \text{Variance}$.

Series completion. This volume completes the five-volume *Mathematics of Neural Networks* series. The series has derived, from first principles, the entire mathematical foundation of modern deep learning:

Vol. I: Differential calculus of neural networks, backpropagation, universal approximation, convolutional weight sharing, and multi-head attention.
Vol. II: Measure-theoretic probability, variational inference (CAVI, ELBO), the EM algorithm, VAEs, normalising flows, GANs, diffusion models, score matching, Gaussian processes, and the generative modelling landscape.
Vol. III: Differential geometry, Riemannian manifolds, group equivariance ($\mathrm{E}(3)$, $\mathrm{SE}(3)$, S_n), graph neural networks, optimal transport, and geometric deep learning.
Vol. IV: Convex and non-convex optimisation, Adam and natural gradient, PAC-Bayes theory, adversarial robustness, Lipschitz networks, neural ODEs, and the adjoint method.
Vol. V: Efficient attention (FlashAttention, SWA, GQA, MLA), state space models (HiPPO, S4, Mamba, RWKV), scaling laws, emergence, mechanistic interpretability (superposition, SAEs, circuits), RLHF/DPO/GRPO, PEFT (LoRA, adapters, prefix tuning), speculative decoding, mixture of depths, and the mathematical frontiers of deep learning.

The series has maintained its four commitments throughout: every derivation step-by-step, every symbol defined, every abstract result grounded in a numerical example, and every geometric interpretation made explicit.

PART 83

Constitutional AI: Self-Critique, Revision and Red-Teaming Mathematics

We formalise the Constitutional AI (CAI) pipeline mathematically, derive the self-critique objective as a contrastive loss, and analyse the convergence of iterative self-improvement under bounded reward model error.

83.1 The Constitutional AI Framework

Constitutional AI (Bai et al. 2022) trains a helpful, harmless, honest (HHH) assistant without relying on human labels of harmful content. The pipeline:

1. *Supervised Learning (SL) phase:* Generate responses to harmful prompts. Self-critique: "Does this response violate principle P?" Self-revise: "Rewrite the response to satisfy P." Fine-tune the model on revised responses.
2. *RL phase:* Train a preference model (PM) using AI-generated harmlessness preferences. Run RLHF against the PM.

Self-critique as a classifier. Let $c_\theta(x, y)$ be the model's critique score for response y to prompt x. The critique is implemented as a prompted inference:

$$c_\theta(x, y) = p_\theta(\text{"harmful"} \mid [\text{critique prompt}] + x + y)$$

The revision model then minimises the critique score while maximising helpfulness:

$$y' = \arg\max_y \lambda_{\text{help}} p_\theta(y|x) - \lambda_{\text{harm}} c_\theta(x, y)$$

In practice, this is approximated by beam search with the combined score.

AI preference model. The AI-generated preference data: $(x, y_{\text{harmless}}, y_{\text{harmful}})$ pairs where y_{harmless} is the revised response and y_{harmful} is the original. The preference model is trained with the standard Bradley-Terry loss:

$$\mathcal{L}_{\text{PM}} = -\log \sigma(r_\phi(x, y_{\text{harmless}}) - r_\phi(x, y_{\text{harmful}}))$$

Constitutional principles as constraints. Let $\mathcal{P} = \{P_1, \ldots, P_K\}$ be a set of constitutional principles. The ideal policy satisfies:

$$\pi^* = \arg\max_\pi \mathbb{E}[r_{\text{help}}(x, \pi(x))] \quad \text{s.t.} \quad \mathbb{E}[c_k(x, \pi(x))] \le \epsilon_k \quad \forall k$$

This is a constrained optimisation. CAI approximates it via Lagrangian relaxation, which is equivalent to the KL-penalised objective when the constraints are soft.

83.2 Iterative Self-Improvement

Iterative CAI

Starting from policy $\pi^{(0)}$, iterate:

1. Generate revisions: $y_i' \sim \pi^{(t)}(\cdot|\text{revise}(x_i, y_i))$.
2. Fine-tune: $\pi^{(t+1)} = \text{SFT}(\pi^{(t)}, \{(x_i, y_i')\})$.

Convergence analysis. Define the "harmlessness gap":

$$\Delta^{(t)} = \mathbb{E}[\text{harmlessness}(\pi^{(t)})] - \mathbb{E}[\text{harmlessness}(\pi^*)]$$

Under the assumption that: (a) the critique is α-accurate (classifies harmful responses correctly with probability $\geq \alpha$), and (b) the revision reduces harm score by β fraction on average, then:

$$\Delta^{(t+1)} \leq (1 - \alpha\beta)\Delta^{(t)}$$

Convergence to π^* requires $\alpha\beta > 0$, i.e. the critique and revision are both effective. The rate $(1 - \alpha\beta)^t$ is geometric, giving linear convergence in the number of self-improvement iterations.

83.3 Red-Teaming as Adversarial Optimisation

Red-teaming generates adversarial prompts that elicit harmful outputs. Formally, the red team solves:

$$x^* = \arg\max_{x} c(x, \pi(x))$$

where c is the harm criterion. This is an adversarial game between the red team (maximising harm) and the policy (minimising harm).

Minimax formulation. The safe policy optimisation:

$$\pi^* = \arg\min_{\pi} \max_{x \in \mathcal{X}} c(x, \pi(x))$$

This is a standard minimax problem. For neural networks, the inner maximisation is non-convex (standard NLP-style adversarial attack); the outer minimisation is the policy training.

Gradient-based red-teaming. Directly differentiate through the critic:

$$\nabla_x c(x, \pi(x)) = \nabla_x c + (\nabla_x \pi(x))^\top \nabla_y c$$

The second term requires differentiating through the policy's generation, which is non-differentiable for discrete outputs. Gumbel-softmax or REINFORCE gradient estimators are used to approximate this.

Part 84

LLM Watermarking: Green-Red Token Lists and Statistical Detection

We derive the Kirchenbauer et al. watermarking scheme for LLMs, prove the statistical power of the z-test detector, and analyse the tradeoff between watermark strength and text quality degradation.

84.1 The Green-Red Watermarking Scheme

Kirchenbauer et al. (2023) embed watermarks by partitioning the vocabulary into "green" and "red" tokens and boosting the logits of green tokens.

Watermarked Sampling

For each position t, given the previous token x_{t-1}:

1. Hash x_{t-1} (plus optional context) to seed a pseudorandom generator.
2. Randomly partition $\mathcal{V} = \mathcal{G}_t \cup \mathcal{R}_t$ with $|\mathcal{G}_t| = \gamma|\mathcal{V}|$.
3. Add bias $\delta > 0$ to logits of tokens in $\mathcal{G}_t$: $f_v^{\text{wm}} = f_v + \delta \cdot \mathbf{1}[v \in \mathcal{G}_t]$.
4. Sample from softmax(f^{wm}).

Green token rate. Without watermark, each token is green with probability γ (the partition fraction). With watermark ($\delta > 0$), green tokens are preferentially sampled: the expected fraction of green tokens in a watermarked text $> \gamma$.

84.2 Statistical Detection

For a text of T tokens, let s_G be the number of green tokens. The null hypothesis H_0 (no watermark): $\mathbb{E}[s_G] = \gamma T$.

z-test statistic.

$$z = \frac{s_G - \gamma T}{\sqrt{T\gamma(1-\gamma)}}$$

Under H_0, $z \sim \mathcal{N}(0,1)$ (by CLT). Reject H_0 (detect watermark) at significance level α when $z > z_\alpha$ (one-sided test).

Statistical power. Let p_G be the true green token rate with watermark. The expected z-score:

$$\mathbb{E}[z] = \frac{(p_G - \gamma)\sqrt{T}}{\sqrt{\gamma(1-\gamma)}}$$

Power approaches 1 as $T \to \infty$ or $(p_G - \gamma) \to \infty$ (strong watermark).

Minimum detectable text length. For power ≥ 0.9 at significance $\alpha = 0.01$ (with $z_\alpha = 2.33$, $z_\beta = 1.28$):

$$T_{\min} = \frac{(z_\alpha + z_\beta)^2 \gamma(1-\gamma)}{(p_G - \gamma)^2} = \frac{(3.61)^2 \gamma(1-\gamma)}{(p_G - \gamma)^2}$$

For $\gamma = 0.5$, $\delta = 2$ (strong watermark), $p_G \approx 0.74$: $T_{\min} = (3.61)^2 \times 0.25/(0.24)^2 \approx 57$ tokens. Detection is possible with as few as ~ 60 tokens.

84.3 Quality Degradation

KL divergence from watermarking. The watermarked distribution q_{wm} vs. the base distribution q:

$$D_{\text{KL}}(q_{\text{wm}} \| q) = \sum_v q_{\text{wm}}(v) \log \frac{q_{\text{wm}}(v)}{q(v)}$$

For $\delta \to 0$ (small bias), Taylor expansion:

$$D_{\text{KL}} \approx \frac{\delta^2}{2} \sum_v q(v)(g_v - \gamma)^2 \approx \frac{\delta^2 \gamma(1-\gamma)}{2}$$

where $g_v = \mathbf{1}[v \in \mathcal{G}]$. The quality degradation is $O(\delta^2)$ while the detection power is $O(\delta)$, so the SNR improves as $\delta \to 0$ (infinitesimal watermarks are undetectable but also detectable with infinite text).

Hard red-list vs. soft bias. Setting $f_v^{\text{wm}} = -\infty$ for red tokens ($\delta \to \infty$) maximises detection power but can catastrophically degrade quality (many valid tokens become impossible). The δ parameter controls the strength-quality tradeoff.

PART 85

Activation Functions: SwiGLU, Mish, and the Smooth Approximation Theory

We derive the SwiGLU activation mathematically, prove that smooth activations provide better gradient flow than ReLU, and analyse the neural tangent kernel for different activation functions.

85.1 SwiGLU Derivation

SwiGLU (Shazeer, 2020) combines a Swish-gated linear unit:

$$\text{SwiGLU}(\mathbf{x}, \mathbf{W}, \mathbf{V}, b, c) = \text{Swish}(\mathbf{x}\mathbf{W} + b) \odot (\mathbf{x}\mathbf{V} + c)$$

where $\text{Swish}(x) = x \cdot \sigma(x)$ and $\odot$ is elementwise multiplication.

Why SwiGLU outperforms ReLU FFN. The ReLU FFN computes:

$$\text{ReLU-FFN}(\mathbf{x}) = \max(0, \mathbf{x}\mathbf{W}_1 + b_1)\mathbf{W}_2 + b_2$$

SwiGLU FFN (standard in LLaMA, Gemma, Mistral):

$$\text{SwiGLU-FFN}(\mathbf{x}) = (\text{Swish}(\mathbf{x}\mathbf{W}_1) \odot \mathbf{x}\mathbf{W}_3)\mathbf{W}_2$$

The gating mechanism $\mathbf{x}\mathbf{W}_3$ provides input-dependent activation: neurons can be "turned off" not just by the activation function but by the gate. This is analogous to LSTM gates controlling information flow.

Smooth gradient. ReLU's gradient is 0 for $x < 0$ (dead neurons). Swish's gradient: $\sigma(x) + x\sigma(x)(1-\sigma(x)) = \sigma(x)(1 + x(1-\sigma(x)))$. At $x = -1$: Swish gradient ≈ 0.27 (non-zero, no dead neurons).

85.2 Activation Function Comparisons

Activation	Formula	Range	Gradient at 0	Dead neurons?
ReLU	$\max(0, x)$	$[0, \infty)$	0 (left)	Yes ($x < 0$)
Leaky ReLU	x if $x > 0$ else αx	$\mathbb{R}$	$\alpha < 1$	No
ELU	x if $x > 0$ else $e^x - 1$	$(-1, \infty)$	1	No
GELU	$x\Phi(x)$	$\mathbb{R}$	0.5	No
Swish	$x\sigma(x)$	$\mathbb{R}$	0.5	No
Mish	$x \tanh(\ln(1 + e^x))$	$\mathbb{R}$	0.6	No
SiLU	$=$ Swish	$\mathbb{R}$	0.5	No

GELU derivation. The Gaussian Error Linear Unit: $\text{GELU}(x) = x \cdot P(X \leq x)$ where $X \sim \mathcal{N}(0, 1)$. At $x \gg 0$: $\text{GELU}(x) \to x$ (linear, like ReLU for positive x). At $x \ll 0$: $\text{GELU}(x) \to 0$ (suppressed, like ReLU). At $x \approx 0$: smooth interpolation.

The stochastic interpretation: GELU is the expected output of a neuron that is set to x with probability $\Phi(x)$ and 0 otherwise — a stochastic neuron with dropout probability $1 - \Phi(x)$.

85.3 Neural Tangent Kernel for Different Activations

The NTK $\Theta(x, x')$ at initialisation depends critically on the activation function through the kernel recursion:

$$\Sigma^{(0)}(x, x') = \frac{1}{d}\mathbf{x}^\top \mathbf{x}', \qquad \Sigma^{(h+1)}(x, x') = \mathbb{E}_{(u,v)\sim\mathcal{N}(0, K^{(h)})}[\phi(u)\phi(v)]$$

where $K^{(h)} = \begin{pmatrix} \Sigma^{(h)}(x,x) & \Sigma^{(h)}(x,x') \\ \Sigma^{(h)}(x',x) & \Sigma^{(h)}(x',x') \end{pmatrix}$.

ReLU kernel. For ReLU: $\mathbb{E}[\phi(u)\phi(v)] = \frac{\sqrt{\Sigma^{(h)}(x,x)\Sigma^{(h)}(x',x')}}{2\pi}(\sin\theta + (\pi-\theta)\cos\theta)$ where $\theta = \arccos\left(\frac{\Sigma^{(h)}(x,x')}{\sqrt{\Sigma^{(h)}(x,x)\Sigma^{(h)}(x',x')}}\right)$.

This is the arc-cosine kernel. It is positive semidefinite and tends to 0 as $\|x - x'\| \to \infty$ (localised kernel).

Smooth activations (GELU, Swish) yield smoother NTKs. The smoothness of the kernel $\Theta(x,x')$ determines the function class of NTK regression. Smoother kernels correspond to RKHS with higher Sobolev regularity — functions that are more interpolating and less extrapolating. This explains empirically why GELU models generalise better than ReLU on out-of-distribution data.

PART 86

Transformers and Formal Languages: Recognisability and Expressibility

We characterise the formal languages that transformers can recognise, analyse the RASP language model as an operational semantics for transformers, and prove that hard attention transformers cannot count.

86.1 RASP: Restricted Access Sequence Processing

Weiss et al. (2021) introduce RASP as a programming language that directly corresponds to transformer computations:

RASP Operations

Select: creates an attention pattern from two sequences Q, K and a predicate ϕ:

$$\text{sel}(Q, K, \phi)[i, j] = \phi(Q_i, K_j) \in \{0, 1\}$$

Aggregate: weighted average under an attention pattern A:

$$\text{agg}(A, V)[i] = \frac{\sum_j A[i, j] \cdot V[j]}{\sum_j A[i, j]}$$

Sequence map (pointwise): $\text{smap}(f)(S)[i] = f(S[i])$.

Counting with RASP. To count occurrences of symbol a up to position i:

1. $A = \text{sel}(\text{pos}, \text{pos}, \leq) \wedge \text{sel}(\text{tokens}, a, =)$ (attend to all a at or before position i).
2. $\text{agg}(A, \mathbf{1}) \cdot i$ (average of 1s times length = count).

This requires *uniform attention* (attending to a varying-size prefix), which requires $O(\log n)$ bits of precision in the attention scores. With hard attention (binary patterns), counting is impossible.

86.2 Formal Language Classes

Theorem (Hahn 2020). Hard-attention transformers (attention patterns are sharp argmax) cannot recognise PARITY or MAJORITY for arbitrary length inputs.

Proof sketch. Hard attention selects one token per head per position. PARITY requires global counting (whether an even number of 1s appear). Any hard-attention transformer's output for position i depends on a fixed, finite set of positions (determined by the attention pattern). For sufficiently long inputs, some positions are unreachable, making global information aggregation impossible.

Soft attention can count. With soft (full) attention: $h_i = \sum_j A_{ij} v_j$, the average $\frac{1}{n} \sum_j v_j$ is computable by setting all attention weights equal ($A_{ij} = 1/n$). This implements a sum/-count operation.

Transformer-recognisable languages. Pérez et al. (2021) show that Transformers with arbitrary precision can simulate Turing machines. This places transformer-recognisable languages at RE (recursively enumerable) — all computable languages. With bounded precision (p-bit floats and L layers): $\text{TC}^0 \subseteq$ expressible $\subseteq$ P (conjectured, not proven; actual

bound is TC^0).

86.3 Implications for In-Context Learning

The RASP characterisation suggests that any algorithm expressible in RASP is learnable in-context by a sufficiently large transformer. The key question: does pretraining learn to implement useful RASP programs?

Evidence for algorithmic learning. Chan et al. (2022) show that in-context learning emerges when training data has:

- "Burstiness": items from the same category appear together (mimicking label information in context).
- "Long-tailed distribution": many rare items (forcing generalisation beyond memorisation).

Under these conditions, the optimal Bayes predictor for next-token prediction implements in-context Bayesian updating — exactly the RASP "select and aggregate" operation for extracting the posterior from the context.

Chain of thought as RASP scratchpad. Wei et al. (2022) show that chain of thought (intermediate reasoning steps) enables solving problems beyond TC^0. The mathematical reason: each chain-of-thought token adds one layer of computation. A chain of length T effectively gives the transformer $L + T$ layers, moving the expressibility from TC^0 toward LOGSPACE or P.

Part 87

Instruction Tuning: FLAN, Super-NaturalInstructions and Task Generalisation

We derive the task-generalisation loss objective for instruction tuning, analyse the scaling of zero-shot performance with the number of fine-tuning tasks, and formalise the relationship between instruction following and meta-learning.

87.1 The Instruction Tuning Objective

Instruction tuning (Wei et al. 2022) fine-tunes a pretrained LLM on a diverse set of T tasks $\{\mathcal{T}_1, \ldots, \mathcal{T}_T\}$, each with natural language instructions and input-output pairs.

Instruction Tuning Loss

$$\mathcal{L}_{\text{IT}} = -\frac{1}{T}\sum_{t=1}^{T}\frac{1}{|\mathcal{D}_t|}\sum_{(x,y)\in\mathcal{D}_t}\sum_{i=1}^{|y|}\log p_\theta(y_i|\text{inst}_t, x, y_{<i})$$

where inst_t is the natural language instruction for task t.

Zero-shot generalisation. After instruction tuning on T_{train} tasks, the model is evaluated on held-out tasks $\mathcal{T}_{\text{test}}$. The key empirical finding (Wei et al. 2022): zero-shot performance on $\mathcal{T}_{\text{test}}$ increases steeply with T_{train} up to ~ 200 tasks, then plateaus.

Why does instruction tuning generalise? The Bayesian meta-learning perspective: the model learns a prior over task structures. Given a new instruction, the model infers which task is being described and applies the appropriate processing. The diversity of training tasks ensures broad coverage of the prior, enabling generalisation.

87.2 Task Embeddings and Clustering

Task embedding. Define the task embedding of $\mathcal{T}_t$ as the mean hidden state of the instruction:

$$\mathbf{e}_t = \frac{1}{|\text{inst}_t|}\sum_{i}\mathbf{h}_i^{(L/2)}(\text{inst}_t)$$

(middle-layer representation of the instruction tokens).

Task cluster structure. Tasks with similar instructions have similar embeddings. Semantic clustering (e.g. k-means on $\{\mathbf{e}_t\}$) recovers natural task categories (NLI, QA, summarisation, classification). Instruction tuning benefits from diversity across clusters: maximising task coverage per unit of compute requires selecting tasks from all clusters.

Optimal task sampling. If the zero-shot loss on new tasks is a function of distance to the nearest training task cluster:

$$\mathcal{L}_{\text{zero-shot}}(\mathcal{T}) \propto \min_t d(\mathbf{e}, \mathbf{e}_t)$$

then the optimal training task set minimises the maximum distance from any possible test task to its nearest training task — a facility location problem. In $\mathbb{R}^d$, the optimal solution is a set of T_{train} points minimising the k-centre cost, achievable by greedy farthest-point sampling.

87.3 Chain of Templates and Prompt Robustness

A key challenge for instruction tuning: sensitivity to prompt phrasing. Two semantically equivalent instructions should produce similar outputs.

Prompt invariance. For two instructions inst and inst$'$ that describe the same task, the model is prompt-invariant if: $\mathrm{TV}(\pi(\cdot|\text{inst}, x), \pi(\cdot|\text{inst}', x)) \leq \epsilon$ for all inputs x.

Template diversity training. FLAN and Tk-Instruct train with multiple templates per task (10–20 per task). This reduces prompt sensitivity because the model cannot overfit to any single phrasing. The gradient update from template diversity:

$$\nabla_\theta \mathcal{L}_{\text{IT}} = \frac{1}{KT} \sum_{t=1}^{T} \sum_{k=1}^{K} \nabla_\theta \mathcal{L}_t^{(k)}$$

where K is the number of templates. The averaged gradient is less sensitive to specific phrasing than any single gradient, improving prompt robustness.

PART 88

Position Encoding: A Complete Survey of Absolute, Relative and Rotary Methods

We derive all major position encoding schemes from a common functional framework, prove their properties under the group structure of sequences, and compare their extrapolation behaviour analytically.

88.1 Functional Framework for Position Encodings

A position encoding assigns to each position $m \in \{0, 1, \ldots, n-1\}$ a vector $\mathbf{p}_m \in \mathbb{R}^d$. The encoded input at position m is $\mathbf{x}_m + \mathbf{p}_m$.

Requirements.

1. *Uniqueness*: different positions have different encodings.
2. *Bounded range*: $\|\mathbf{p}_m\|$ should not grow unboundedly with m.
3. *Relative sensitivity*: nearby positions should have similar encodings; distant ones, different.
4. *Extrapolation*: the encoding should behave consistently for positions beyond the training range.

Sinusoidal (Vaswani et al. 2017).

$$p_{m,2i} = \sin(m/10000^{2i/d}), \quad p_{m,2i+1} = \cos(m/10000^{2i/d})$$

Satisfies 1 (unique), 2 (bounded to $[-1, 1]$), 3 (nearby positions similar), 4 (extrapolates by periodicity). Problem: inner product $\mathbf{p}_m^\top \mathbf{p}_n$ does not depend only on $m - n$ — not purely relative.

88.2 Relative Position Encodings

T5 relative bias (Raffel et al. 2020). Add a scalar $b_{ij} = r(i-j)$ to attention logits:

$$A_{ij} = \text{softmax}_j\left(\frac{q_i^\top k_j}{\sqrt{d_k}} + b_{ij}\right)$$

where $r : \mathbb{Z} \to \mathbb{R}$ is a learned embedding with shared values for large $|i-j|$. Satisfies relative sensitivity and is position-translation-invariant.

ALiBi (Press et al. 2022). Linear bias with head-specific slopes:

$$b_{ij}^{(h)} = -m_h \cdot (i-j), \quad m_h = 2^{-8h/H}$$

Slopes are fixed (not learned). ALiBi's linear decay encourages the model to attend to nearby tokens and discourages attending to distant ones, providing built-in locality bias. Extrapolation: for positions $i - j > n_{\text{train}}$, the penalty $-m_h(i-j)$ continues growing, softly discouraging attention to very distant tokens.

88.3 RoPE: Full Derivation from Requirements

Design requirement. We want a function $f(q, m)$ (encoding position m into query q) such that:

$$\langle f(q,m), f(k,n)\rangle = g(q,k,m-n)$$

for some function g depending only on the relative position $m-n$.

Solution in 2D. For $d=2$, the requirement becomes: $\langle \mathbf{R}_m q, \mathbf{R}_n k\rangle = q^\top \mathbf{R}_n^\top \mathbf{R}_m k = q^\top \mathbf{R}_{n-m} k$, which holds for any rotation matrix $\mathbf{R}_\theta$ since $\mathbf{R}_{m-n}^\top = \mathbf{R}_{n-m}$.

Generalisation to d dimensions. Partition the d dimensions into $d/2$ pairs. Apply a 2D rotation to each pair with a distinct frequency θ_i:

$$f(\mathbf{q}, m) = \begin{pmatrix} q_1 \cos m\theta_1 - q_2 \sin m\theta_1 \\ q_1 \sin m\theta_1 + q_2 \cos m\theta_1 \\ \vdots \\ q_{d-1} \cos m\theta_{d/2} - q_d \sin m\theta_{d/2} \\ q_{d-1} \sin m\theta_{d/2} + q_d \cos m\theta_{d/2} \end{pmatrix}$$

The inner product then satisfies the relative position requirement, since each 2D block contributes $q_{2i-1}k_{2i-1}\cos(m-n)\theta_i + q_{2i}k_{2i}\cos(m-n)\theta_i + \ldots$ — a function of $(m-n)$ only.

Frequency choice. RoPE sets $\theta_i = 10000^{-2(i-1)/d}$ (geometric sequence from 1 to 10000^{-1}). This gives wavelengths from 2π (pair 1, high frequency) to $2\pi \times 10000$ (pair $d/2$, low frequency), covering five orders of magnitude. Information at each scale is captured by the corresponding dimension pair.

88.4 Comparison: Extrapolation Properties

Method	Relative?	Extrapolation	Memory cost
Sinusoidal	Partial	Periodic (limited)	$O(nd)$
Learned Abs	No	Poor (undefined)	$O(n_{\text{max}}d)$
T5 Relative	Yes	Good (shared params)	$O(n^2)$ attn
ALiBi	Yes	Excellent (continues linearly)	$O(n^2)$ attn
RoPE	Yes	Good (periodic, well-calibrated)	$O(nd)$

Why RoPE extrapolates better than sinusoidal. Both are periodic, but RoPE's inner product depends only on the *relative* rotation $\mathbf{R}_{m-n}$, not absolute positions. A model trained on relative positions $\{0, \ldots, n_{\text{train}}\}$ sees each relative gap $m-n$ uniformly; at test time, seeing gap $m-n = n_{\text{test}}$ requires only that the corresponding rotation is well-defined, which it is for all $(m-n) \in \mathbb{Z}$.

Appendix A: Notation and Symbols

The following conventions are used uniformly throughout Volume V.

A.1 Scalars, Vectors, Matrices, Tensors

Symbol	Type	Description
a, b, c	scalar	lowercase italic
$\mathbf{v}, \mathbf{q}, \mathbf{k}$	column vector	bold lowercase
$\mathbf{A}, \mathbf{W}, \mathbf{K}$	matrix	bold uppercase
$\mathcal{T}$	tensor (order ≥ 3)	calligraphic
$\mathbf{I}_d$	$d \times d$ identity	standard
$\mathbf{0}_{m\times n}$	zero matrix	standard
$\mathbf{A}^\top$	transpose	
$\mathbf{A}^{-1}$	inverse (square)	
$\mathbf{A}^+$	Moore–Penrose pseudoinverse	
$\|\mathbf{v}\|_2$	Euclidean norm	
$\|\mathbf{A}\|_F$	Frobenius norm	$\sqrt{\sum_{ij} A_{ij}^2}$
$\|\mathbf{A}\|_*$	nuclear norm	sum of singular values
$\langle \mathbf{u}, \mathbf{v} \rangle$	inner product	$\mathbf{u}^\top \mathbf{v}$

A.2 Probability and Distributions

Symbol	Meaning	Notes
$p(\cdot), q(\cdot)$	probability densities / PMFs	
P, Q	probability measures	
$\mathbb{E}_{x\sim p}[\cdot]$	expectation under p	abbreviated $\mathbb{E}[\cdot]$ when clear
$\mathrm{Var}[\cdot]$	variance	
$D_{\mathrm{KL}}(P\|Q)$	KL divergence	$\int p \log(p/q)$
$H(P)$	Shannon entropy	$-\int p \log p$
$H(P, Q)$	cross-entropy	$-\int p \log q$
$\mathcal{N}(\mu, \Sigma)$	Gaussian	
$\mathrm{Cat}(\pi)$	categorical	
$\mathrm{Ber}(\rho)$	Bernoulli	
$\sigma(\cdot)$	sigmoid	$(1 + e^{-(\cdot)})^{-1}$
$\mathrm{softmax}(\mathbf{z})_i$		$e^{z_i} / \sum_j e^{z_j}$

A.3 Model-Specific Notation

Symbol	Meaning	First introduced
d, d_k, d_v	embedding / key / value dimension	§53
H	number of attention heads	§53
n, L	sequence length / context window	§53
$\mathbf{Q}, \mathbf{K}, \mathbf{V}$	query, key, value matrices	§53
$\mathbf{R}_m$	RoPE rotation matrix at position m	§54
$\mathbf{A}, \mathbf{B}, \mathbf{C}$	SSM state, input, output matrices	§55
$\bar{\mathbf{A}}, \bar{\mathbf{B}}$	discretised SSM matrices	§55
Δ	SSM discretisation step	§55
r	LoRA rank	§66
α_{LoRA}	LoRA scaling factor	§66
β	alignment KL penalty	§62
N, D	model parameters / training tokens (scaling)	§57
C	FLOPs budget	§57

A.4 Complexity and Asymptotic Notation

- $O(f)$: asymptotic upper bound (big-O).
- $\Omega(f)$: asymptotic lower bound.
- $\Theta(f)$: tight asymptotic bound.
- $\tilde{O}(f)$: big-O hiding logarithmic factors.
- $\text{poly}(n)$: some fixed polynomial in n.

Appendix B: Key Derivations

B.1 Softmax Attention as a Kernel Regression

Let $\mathbf{Q} \in \mathbb{R}^{n\times d_k}$, $\mathbf{K}, \mathbf{V} \in \mathbb{R}^{m\times d_k}$ and $\mathbf{V} \in \mathbb{R}^{m\times d_v}$. Define the kernel $\kappa(\mathbf{q}, \mathbf{k}) = \exp(\mathbf{q}^\top \mathbf{k}/\sqrt{d_k})$. Then the attention output for query $\mathbf{q}_i$ is:

$$\mathrm{Attn}(\mathbf{q}_i) = \frac{\sum_{j=1}^{m} \kappa(\mathbf{q}_i, \mathbf{k}_j)\, \mathbf{v}_j}{\sum_{j=1}^{m} \kappa(\mathbf{q}_i, \mathbf{k}_j)} = \sum_{j=1}^{m} w_{ij}\, \mathbf{v}_j,$$

where $w_{ij} = \kappa(\mathbf{q}_i, \mathbf{k}_j) \big/ \sum_l \kappa(\mathbf{q}_i, \mathbf{k}_l)$. This is the Nadaraya–Watson estimator with exponential kernel, placing attention in the framework of non-parametric kernel regression.

B.2 FlashAttention: Exact Online Softmax

Online softmax avoids materialising the full $n \times n$ attention matrix. For a block of keys $\mathbf{K}_s$ and the running maximum $m^{(s)}$, the recurrence is:

$$m^{(s+1)} = \max\Big(m^{(s)},\ \max_j\Big(\mathbf{q}^\top \mathbf{k}_j^{(s+1)}/\sqrt{d_k}\Big)\Big) \tag{1}$$

$$\ell^{(s+1)} = e^{m^{(s)} - m^{(s+1)}} \ell^{(s)} + \sum_j e^{s_j^{(s+1)} - m^{(s+1)}} \tag{2}$$

$$\mathbf{o}^{(s+1)} = e^{m^{(s)} - m^{(s+1)}} \mathbf{o}^{(s)} + \sum_j e^{s_j^{(s+1)} - m^{(s+1)}} \mathbf{v}_j^{(s+1)}, \tag{3}$$

where $s_j^{(s)} = \mathbf{q}^\top \mathbf{k}_j^{(s)}/\sqrt{d_k}$. After all blocks: output $= \mathbf{o}^{(S)}/\ell^{(S)}$. IO complexity is $O(n^2 d/M)$ for SRAM of size M, versus $O(n^2)$ for naive attention.

B.3 Chinchilla: Derivation of Optimal Token Count

The Hoffmann et al. (2022) loss surface is modelled as:

$$L(N, D) = E + \frac{A}{N^\alpha} + \frac{B}{D^\beta},$$

with fitted constants $E, A, B, \alpha, \beta > 0$. Under a compute budget $C = 6ND$ (FLOPs for one forward+backward pass on D tokens with N parameters), we minimise L subject to $6ND = C$:

$$\frac{\partial L}{\partial N} = -\frac{A\alpha}{N^{\alpha+1}} + \lambda \cdot 6D = 0 \tag{4}$$

$$\frac{\partial L}{\partial D} = -\frac{B\beta}{D^{\beta+1}} + \lambda \cdot 6N = 0. \tag{5}$$

Eliminating λ gives:

$$\frac{A\alpha}{N^{\alpha+1}} \cdot 6N = \frac{B\beta}{D^{\beta+1}} \cdot 6D \implies \frac{A\alpha}{N^\alpha} = \frac{B\beta}{D^\beta}.$$

The fitted exponents $\alpha \approx \beta$ imply $N \propto D$, i.e. parameters and tokens should scale proportionally. The Chinchilla-optimal ratio is approximately $D^* \approx 20N$.

B.4 RLHF: PPO Clipped Objective

The Proximal Policy Optimisation (PPO) clipped objective for language model alignment is:

$$\mathcal{L}^{\text{PPO}}(\theta) = \mathbb{E}_t\Big[\min\Big(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t\Big)\Big]\,,$$

where $r_t(\theta) = \pi_\theta(a_t|s_t)/\pi_{\theta_{\text{old}}}(a_t|s_t)$ is the probability ratio and $\hat{A}_t$ is the generalised advantage estimate. The full LM PPO objective adds a KL penalty against the reference policy π_{ref}:

$$\mathcal{L}(\theta) = \mathcal{L}^{\text{PPO}}(\theta) - \beta\, D_{\text{KL}}(\pi_\theta \| \pi_{\text{ref}}).$$

B.5 DPO: Closed-Form Policy from Bradley–Terry

Starting from the Bradley–Terry preference model $p(y_w \succ y_l \mid x) = \sigma(r(x, y_w) - r(x, y_l))$ and the reward parameterisation $r(x, y) = \beta \log \frac{\pi_\theta(y|x)}{\pi_{\text{ref}}(y|x)} + \beta \log Z(x)$, the partition function $Z(x)$ cancels in the preference probability, giving:

$$p_\theta(y_w \succ y_l \mid x) = \sigma\left(\beta \log \frac{\pi_\theta(y_w|x)}{\pi_{\text{ref}}(y_w|x)} - \beta \log \frac{\pi_\theta(y_l|x)}{\pi_{\text{ref}}(y_l|x)}\right).$$

The DPO loss is the negative log-likelihood under this model over a preference dataset $\mathcal{D} = \{(x, y_w, y_l)\}$, which can be minimised directly without training a separate reward model.

Appendix D: Glossary of Algorithms

A concise reference for the algorithms, architectures, and methods introduced in Volume V, ordered by first appearance.

GQA *Grouped Query Attention* (§53.3). Reduces KV-cache by sharing key/value heads across groups of query heads. Interpolates between multi-head attention (one group per head) and multi-query attention (one group total).

MoE *Mixture-of-Experts* (§53.4). Replaces a dense FFN with E expert networks; a router selects the top-k experts per token. Increases model capacity without proportional compute.

RoPE *Rotary Positional Embedding* (§54.1). Encodes position by rotating query/key vectors in 2D subspaces; inner products depend only on relative position.

YaRN *Yet another RoPE extensioN* (§54.2). Extends RoPE context length by applying frequency-dependent interpolation and temperature scaling.

ALiBi *Attention with Linear Biases* (§54.3). Adds a position-dependent linear penalty to attention logits; no positional embeddings required.

S4 *Structured State Space Sequence model* (§55.2). Uses a diagonal-plus-low-rank (DPLR) parameterisation of the SSM state matrix $\mathbf{A}$ to enable efficient $O(n \log n)$ convolution.

Mamba *Selective State Space Model* (§55.3). Extends S4 with input-dependent $\mathbf{B}, \mathbf{C}, \Delta$ matrices; enables selective state updates. Linear recurrence at inference; parallel scan at training.

FAVOR+ *Fast Attention Via positive Orthogonal Random features* (§53.5). Approximates softmax attention as a low-rank outer-product sum using random feature maps, achieving $O(n)$ attention.

HiPPO *High-order Polynomial Projection Operator* (§55.4). Derives the optimal SSM matrix $\mathbf{A}$ for online polynomial approximation of input history.

SAE *Sparse Autoencoder* (§61.2). Dictionary learning model trained to reconstruct activations with a sparse bottleneck; used to decompose polysemantic neurons into monosemantic features.

DPO *Direct Preference Optimisation* (§62). Trains a preference-aligned policy without an explicit reward model, using a closed-form Bradley–Terry objective.

GRPO *Group Relative Policy Optimisation* (§62). Variant of PPO that estimates advantages from a group of sampled completions, eliminating the critic network.

DARTS *Differentiable ARchiTecture Search* (§64.2). Relaxes NAS over a categorical architecture distribution; optimises architecture weights with gradient descent.

LoRA *Low-Rank Adaptation* (§66.2). Freezes pretrained weights and adds trainable rank-r decomposition $\mathbf{W}+\mathbf{BA}$; reduces trainable parameters from $O(d^2)$ to $O(dr)$.

QLoRA *Quantised LoRA* (§66.3). Combines 4-bit NF4 quantisation of frozen weights with LoRA adapters and double quantisation; enables LLM fine-tuning on consumer GPU.

TTT *Test-Time Training* (§66.4). Updates model weights at inference on a self-supervised task constructed from the input context; effectively uses test examples to adapt the hidden state.

CLIP *Contrastive Language-Image Pre-training* (§67.1). Trains dual encoder (image/-text) with contrastive loss on image-caption pairs; aligns vision and language representations.

ViT *Vision Transformer* (§67.2). Treats image as a sequence of flattened patch embeddings processed by standard transformer encoder.

ToT *Tree of Thoughts* (§59.4). Structured search over a tree of intermediate reasoning steps; nodes evaluated by a critic; supports BFS/DFS.

CAI *Constitutional AI* (§83). Self-critique and revision pipeline guided by a written constitution; produces harmless outputs without human red-teaming labels.

RASP *Restricted Access Sequence Processing* (§86.1). Domain-specific language for programs expressible by transformers; characterises the formal language class computable by constant-depth attention.

Appendix E: Complexity Reference Table

Time and space complexities for the principal architectures and algorithms in this volume. n = sequence length, d = model dimension, H = heads, E = experts, k = top-k, r = LoRA rank, M = SRAM size.

Method	Time	Memory	Notes
Standard MHA	$O(n^2 d)$	$O(n^2)$	KV-cache $O(nd)$ per layer
MQA	$O(n^2 d)$	$O(nd)$	single KV head
GQA (g groups)	$O(n^2 d)$	$O(ng\,d/H)$	interpolates MHA/MQA
FlashAttention	$O(n^2 d)$	$O(n)$	IO-optimal for SRAM M
FAVOR+	$O(nd\,r_f)$	$O(nr_f)$	r_f random features
Linear attention	$O(nd^2)$	$O(d^2)$	causal; no n^2 term
S4 (conv)	$O(n \log n)$	$O(n)$	FFT-based
Mamba (train)	$O(n \log n)$	$O(n)$	parallel scan
Mamba (infer)	$O(d_{\text{state}})$ per step	$O(d_{\text{state}})$	pure recurrence
Sparse MoE (E, top-k)	k/E of dense FFN	same	routing overhead
LoRA fine-tuning	$O(ndr)$	$O(nr + r^2)$	$r \ll d$
DPO	same as SFT	same	no reward model
DARTS (outer loop)	$O(\lvert\alpha\rvert \cdot \text{fwd})$	$O(\lvert\alpha\rvert)$	bilevel

Parallelism modes.

- *Tensor parallelism*: split $\mathbf{W}$ column/row-wise across devices; communication at each layer.
- *Pipeline parallelism*: partition layers across devices; micro-batch scheduling to reduce bubble.
- *Sequence parallelism*: partition the n dimension across devices; requires all-gather on activations.
- *Expert parallelism*: each device hosts a subset of MoE experts; all-to-all routing.

Bibliography

The following references are the primary sources for the material in Volume V. Author–year format; entries ordered alphabetically by first author.

Ainslie, J., Lee-Thorp, J., de Jong, M., Zeiler, M., Sanghai, S., & Xu, Y. (2023). GQA: Training generalised multi-query transformer models from multi-head checkpoints. *arXiv:2305.13245*.

Anil, R., et al. (2023). PaLM 2 technical report. *arXiv:2305.10403*.

Artetxe, M., Bhosale, S., Goyal, N., et al. (2021). Efficient large scale language modeling with mixtures of experts. *arXiv:2112.10684*.

Bai, Y., Jones, A., Ndousse, K., et al. (2022). Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv:2204.05862*.

Bai, Y., Kadavath, S., Kundu, S., et al. (2022). Constitutional AI: Harmlessness from AI feedback. *arXiv:2212.08073*.

Brown, T., Mann, B., Ryder, N., et al. (2020). Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33, 1877–1901.

Chen, M., Tworek, J., Jun, H., et al. (2021). Evaluating large language models trained on code. *arXiv:2107.03374*.

Child, R., Gray, S., Radford, A., & Sutskever, I. (2019). Generating long sequences with sparse transformers. *arXiv:1904.10509*.

Choromanski, K., Likhosherstov, V., Dohan, D., et al. (2021). Rethinking attention with Performers. *International Conference on Learning Representations*.

Clark, A., de las Casas, D., Guy, A., et al. (2022). Unified scaling laws for routed language models. *arXiv:2202.01169*.

Dao, T., Fu, D. Y., Ermon, S., Rudra, A., & Ré, C. (2022). FlashAttention: Fast and memory-efficient exact attention with IO-awareness. *Advances in Neural Information Processing Systems*, 35.

Dao, T. (2023). FlashAttention-2: Faster attention with better parallelism and work partitioning. *arXiv:2307.08691*.

Dettmers, T., Pagnoni, A., Holtzman, A., & Zettlemoyer, L. (2023). QLoRA: Efficient finetuning of quantised LLMs. *Advances in Neural Information Processing Systems*, 36.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., et al. (2021). An image is worth 16×16 words: Transformers for image recognition at scale. *International Conference on Learning Representations*.

Fedus, W., Zoph, B., & Shazeer, N. (2022). Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120), 1–39.

Geva, M., Schuster, R., Berant, J., & Levy, O. (2021). Transformer feed-forward layers are key-value memories. *Empirical Methods in Natural Language Processing*.

Graves, A. (2016). Adaptive computation time for recurrent neural networks. *arXiv:1603.08983*.

Guo, D., Zhu, Q., Yang, D., et al. (2024). DeepSeek-R1: Incentivising reasoning capability in LLMs via reinforcement learning. *arXiv:2401.12948*.

Gu, A., & Dao, T. (2023). Mamba: Linear-time sequence modeling with selective state spaces. *arXiv:2312.00752*.

Gu, A., Goel, K., & Ré, C. (2022). Efficiently modeling long sequences with structured state spaces. *International Conference on Learning Representations*.

Gu, A., Johnson, I., Goel, K., et al. (2020). HiPPO: Recurrent memory with optimal polynomial projections. *Advances in Neural Information Processing Systems*, 33.

He, J., Zhou, C., Ma, X., et al. (2022). Towards a unified view of parameter-efficient transfer learning. *International Conference on Learning Representations*.

Hoffmann, J., Borgeaud, S., Mensch, A., et al. (2022). Training compute-optimal large language models. *arXiv:2203.15556*.

Hu, E. J., Shen, Y., Wallis, P., et al. (2022). LoRA: Low-rank adaptation of large language models. *International Conference on Learning Representations*.

Jacobs, R., Jordan, M., Nowlan, S., & Hinton, G. (1991). Adaptive mixtures of local experts. *Neural Computation*, 3(1), 79–87.

Kaplan, J., McCandlish, S., Henighan, T., et al. (2020). Scaling laws for neural language models. *arXiv:2001.08361*.

Lieber, O., Sharir, O., Lenz, B., & Shoham, Y. (2021). Jurassic-1: Technical details and evaluation. Technical report, AI21 Labs.

Liu, H., Li, C., Wu, Q., & Lee, Y. J. (2024). Visual instruction tuning (LLaVA). *Advances in Neural Information Processing Systems*, 36.

Liu, S., et al. (2023). Ring attention with blockwise transformers for near-infinite context. *arXiv:2310.01889*.

Meta AI. (2023). Llama 2: Open foundation and fine-tuned chat models. *arXiv:2307.09288*.

Micikevicius, P., Narang, S., Alben, J., et al. (2018). Mixed precision training. *International Conference on Learning Representations*.

Mnih, V., Kavukcuoglu, K., Silver, D., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518, 529–533.

Olah, C., Cammarata, N., Schubert, L., et al. (2020). Zoom in: An introduction to circuits. *Distill*. https://distill.pub/2020/circuits/zoom-in/

Peng, B., et al. (2023). RWKV: Reinventing RNNs for the transformer era. *arXiv:2305.13048*.

Press, O., Smith, N. A., & Lewis, M. (2022). Train short, test long: Attention with linear biases enables input length extrapolation. *International Conference on Learning Representations*.

Radford, A., Wu, J., Child, R., et al. (2019). Language models are unsupervised multitask learners. Technical report, OpenAI.

Rafailov, R., Sharma, A., Mitchell, E., et al. (2023). Direct preference optimisation: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimisation algorithms. *arXiv:1707.06347*.

Shannon, C. E. (1948). A mathematical theory of communication. *Bell System Technical Journal*, 27(3), 379–423.

Shazeer, N. (2020). GLU variants improve transformer. *arXiv:2002.05202*.

Shazeer, N., Mirhoseini, A., Maziarz, K., et al. (2017). Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *International Conference on Learning Representations*.

Su, J., Lu, Y., Pan, S., Wen, B., & Liu, Y. (2024). RoFormer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568, 127063.

Sun, Y., Dong, L., Huang, S., et al. (2023). Retentive network: A successor to transformer for large language models. *arXiv:2307.08621*.

Tolstikhin, I., Houlsby, N., Kolesnikov, A., et al. (2021). MLP-Mixer: An all-MLP architecture for vision. *Advances in Neural Information Processing Systems*, 34.

Touvron, H., Martin, L., Stone, K., et al. (2023). Llama 2: Open foundation and fine-tuned chat models. *arXiv:2307.09288*.

Vaswani, A., Shazeer, N., Parmar, N., et al. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, 30.

Wang, B., & Komatsuzaki, A. (2021). GPT-J-6B: A 6 billion parameter autoregressive language model. Technical report, EleutherAI.

Wang, S., Li, B. Z., Khabsa, M., Fang, H., & Ma, H. (2020). Linformer: Self-attention with linear complexity. *arXiv:2006.04768*.

Wei, J., Wang, X., Schuurmans, D., et al. (2022). Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35.

Wei, J., Tay, Y., Bommasani, R., et al. (2022). Emergent abilities of large language models. *Transactions on Machine Learning Research*.

Wortsman, M., Ilharco, G., Gadre, S. Y., et al. (2022). Model soups: Averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. *International Conference on Machine Learning*.

Xiao, G., Lin, J., Seznec, M., et al. (2023). SmoothQuant: Accurate and efficient post-training quantisation for large language models. *International Conference on Machine Learning*.

Yao, S., Yu, D., Zhao, J., et al. (2023). Tree of Thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36.

Zhang, S., Roller, S., Goyal, N., et al. (2022). OPT: Open pre-trained transformer language models. *arXiv:2205.01068*.

Ziegler, D. M., Stiennon, N., Wu, J., et al. (2019). Fine-tuning language models from human preferences. *arXiv:1909.08593*.

Zoph, B., & Le, Q. V. (2017). Neural architecture search with reinforcement learning. *International Conference on Learning Representations*.